

Article

Not peer-reviewed version

---

# Cyber Security on the Edge: Efficient Enabling of Machine Learning on IoT Devices

---

[Swati Kumari](#) , Vatsal Tulshyan , [Hitesh Tewari](#) \*

Posted Date: 8 January 2024

doi: 10.20944/preprints202401.0555.v1

Keywords: IoT; Cyber Threats; Distributed Computing; AI-enabled Chips; Container Orchestration; DDoS Attacks




Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

## Article

# Cyber Security on the Edge: Efficient Enabling of Machine Learning on IoT Devices

Swati Kumari <sup>1,†,‡</sup> , Vatsal Tulshyan <sup>2,‡</sup> and Hitesh Tewari <sup>2,\*</sup>

<sup>1</sup> Affiliation 1; kumaris@tcd.ie

<sup>2</sup> Affiliation 2; vatsaltulshyan@gmail.com

<sup>2</sup> Affiliation 3; hitesh.tewari@tcd.ie

\* Correspondence: hitesh.tewari@tcd.ie; Tel.: +353-872122008 (H.T.)

† Current address: Trinity College Dublin.

‡ These authors contributed equally to this work.

**Abstract:** Due to rising cyber threats, IoT devices' security vulnerabilities are expanding. However, these devices cannot run complicated security algorithms locally due to hardware restrictions. Data must be transferred to cloud nodes for processing, giving attackers an entry point. This research investigates distributed computing on the edge, using AI-enabled IoT devices and container orchestration tools to process data in real time at the network edge. The purpose is to identify and mitigate DDoS assaults while minimizing CPU usage to improve security. It compares typical IoT devices with and without AI-enabled chips, container orchestration, and assesses their performance in running machine learning models with different cluster settings. The proposed architecture aims to empower IoT devices to process data locally, minimizing the reliance on cloud transmission and bolstering security in IoT environments. The results correlate with the update in the architecture. With the addition of AI-enabled IoT device and container orchestration, there is a difference of 60% between the new architecture and traditional architecture where only Raspberry Pi were being used.

**Keywords:** IoT; Cyber Threats; Distributed Computing; AI-enabled Chips; Container Orchestration; DDoS Attacks

## 1. Introduction

### 1.1. Background

Since cyber security threats are rising, securing the Internet of Things (IoT) is crucial to daily life. Since IoT hardware cannot manage sophisticated workloads, they cannot run complex security risk detection algorithms on themselves, making them vulnerable to high-level security attacks. Instead, they must send data to cloud nodes to process inputs, leaving a loophole for attackers to steal data midway. However, if they could run those models, they could process their real-time packets or inputs. Security is still important when embedded technology advances and becomes robust enough to create apps on it to make life easier. Security adaptation cannot keep up with rapid technology change. Growing IoT devices have caused exponential growth. Imagine its economic impact on global markets. Distributed computing on the edge is desired to offload activities and improve resource planning and IoT device use due to cloud reliability, latency, and privacy problems.[1]

Most workloads are in the cloud, making data transit vulnerable to hackers. Fitness trackers broadcast important data like heart rate and blood pressure to the cloud, which might be compromised during transmission [2,3]. To circumvent these issues, compute close to IoT devices to scale edge computing and make them capable of processing large amounts of data.

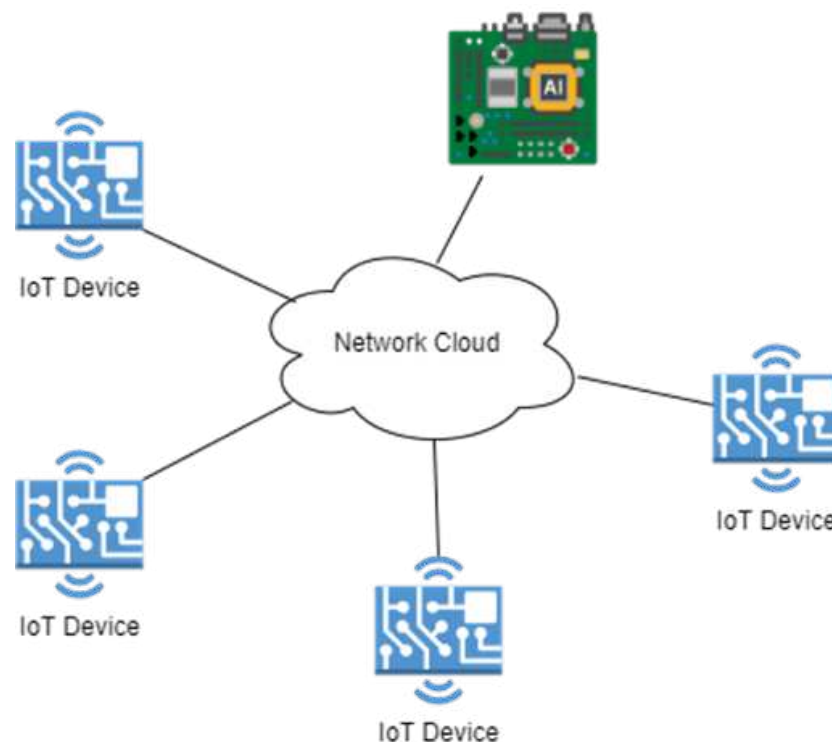
Container orchestration is essential for IoT application management and deployment. Developers may automate container deployment, scaling, and administration to optimize IoT device resources. The dynamic and resource-constrained nature of IoT settings requires load balancing, auto-scaling, and

fault tolerance, which container orchestration provides. IoT service dependability and availability are improved, updates and maintenance are simplified, and developers can focus on developing creative, robust solutions that respond to changing conditions in real time [4].

The methodology involves setting up IoT devices with static IP addresses, building a machine learning model for DDoS detection, and setting up traditional devices (e.g. Raspberry Pi) in a distributed computing environment with an AI-enabled chip IoT device and a container orchestration tool. A Raspberry Pi-only distributed computing cluster with and without container orchestration tools would be compared to this cluster. All clusters will be evaluated by running ML models and comparing their performance and hardware metrics (CPU and Memory).

This research aims to find whether scaling up traditional IoT devices with AI-enabled chips can help with running complex models and whether lightweight orchestration tools could help the resource utilisation on IoT devices and help them to run machine learning and apply them to cyber security improvement.

Figure 1 shows the proposed state-of-the-art architecture in this research. It combines AI-enabled IoT devices with other IoT devices to form a cluster. This design would enable near-edge computation by sending IoT packets to the AI IoT device for processing to detect anomalies. The cloud would preserve these anomalies for future improvements. The figure's network cloud is any useable network interface such as WiFi or swap networks. Due to the AI IoT device's sole purpose of processing and communicating with the cloud, it remains a secure route to transmit information. With the help of container orchestration tools and AI IoT Devices in a networking architecture, the IoT devices will be able to run machine learning models efficiently on the edge to tackle attacks like DDoS by reducing CPU utilization.



**Figure 1.** General Architecture

### 1.2. Research Contribution

This research aims to scale up the performance of IoT devices to support machine learning models and perform security risk detections on top of them with the help of AI-enabled IoT device and container orchestration like Microk8s. This research accomplished its goals by employing the following methods:

1. Use of AI-enabled architecture in the cluster
2. Using the Machine learning models for the detection
3. Use of container orchestration tool like Microk8s
4. Different forms of architecture with AI-enabled IoT devices or simple IoT devices like Raspberry Pi.
5. Docker images build for microservices
6. An overall decrease of 60% in CPU utilisation from the traditional architecture of Raspberry Pi to Microk8s architecture with Nvidia is achieved.
7. Container Orchestration-as-a-solution: It provides a way to efficiently manage the resources, auto-scaling the resources if required and did a great job with scheduling jobs. For the fact, it is security reliable as well since the solutions are not directly deployed to the host therefore any malware implantation can be stopped just by restarting another pod for that particular deployment.

## 2. Related Work

### 2.1. Cyber Security as a Challenge to IoT Devices

IoT devices are being used in many fields. This covers manufacturing, logistical, medical, military, etc. Research predicts 100 billion devices will be in use by 2025 [5]. Considering the predicted amount of devices [5], it's clear that attackers are targeting the IoT for compromise. The attackers would compromise selected nodes in the architecture to access the source code and infect the remainder of the deployments, including cloud resources. IoT deployments have violated local laws in earlier study. Current IoT security methods often rely on manual intervention for maintenance and updates, leading to a lag in protection and an inability to learn and adapt to evolving threats. The unity and integrity of IoT, which spans terminals, networks, and service platforms, necessitate security solutions capable of effectively handling massive amounts of complex data [6].

Traditional data transfer to central servers is still popular. The data is usually sent over MQTT or HTTP. Previous research has shown transmission problems with MQTT and HTTP protocols [7,8]. Attackers find transfer scenarios the entryway to their target information. They would use man-in-the-middle attacks to infect routers that could sneak malicious packets to the main servers. The attackers also turned IoT devices in the large-scale infrastructure into botnets by planting malicious malware. After then, these devices send simultaneous requests to primary servers, causing a Distributed Denial of Service on cloud servers. If the main servers cannot receive data from legitimate IoT devices, incorrect or timed-out responses during data transmission cause data overload. In such a scenario, the data starts collecting and rendering the legitimate devices to shut down operations [9].

### 2.2. Solutions Implemented to Solve Security Vulnerabilities

Malicious attacks have taken many forms. Attackers know that IoT security breaches take time to detect. Due to hardware issues, botnet and malware detection in the IoT environment might be difficult. However, edge computing with machine learning models on IoT devices has yielded novel solutions. Edge computing is a computing method in which the workload is divided amongst nodes to form a distributed architecture. It is an important area in research for IoT as this minimises the bandwidth and response time in an IoT environment. Moreover, it reduces the burden of a centralized server.[10] There have been detection methods which have been developed to be used in an edge computing scenario.

Myneni *et al.* [11] suggests "SmartDefense", a two-stage DDoS detection solution that utilizes deep learning algorithms and operates on the provider edge (PE) and consumer edge (CE). PE and CE refer to the routers which are close to consumers and Internet service providers (ISPs). Edge computing helps SmartDefense detect and stop DDoS attacks near their source, reducing bandwidth waste and provider edge delay. The method also uses a botnet detection engine to detect and halt bot traffic.

SmartDefense improves DDoS detection accuracy and reduces ISP overhead, according to the study. By detecting botnet devices and mitigating over 90% of DDoS traffic coming from the consumer edge, it can cut DDoS traffic by up to 51.95%.

Bhardwaj *et al.* [12] suggests a method called ShadowNet that makes use of computational capabilities at the network's edge to speed up a defence against IoT-DDoS attacks. The edge tier, which includes fog computing gateways and mobile edge computing (MEC) access points, can manage IoT devices' massive Internet traffic using edge services. As IoT packets pass through gateways, edge functions build lightweight information profiles that are quickly acquired and sent to ShadowNet web services. IoT-DDoS attacks are initially prevented by ShadowNet's analysis of edge node data. It detects IoT-DDoS attacks 10 times faster than victim-based methods. It can also react to an attack proactively, stopping up to 82% of the malicious traffic from getting to the target and harming the Internet infrastructure.

Mirzai *et al.* [13] discovers the benefits of building a dynamic user-level scheduler to perform real-time updates of machine learning models and analyse the performance of hardware of Nvidia Jetson Nano and Raspberry Pi. Schedules various models on both devices and analyzes bottlenecks. The scheduler allows local parallel model retraining on the IoT device without stopping the IDS, eliminating cloud resources. The Nvidia unit could retrain models while detecting anomalies, and the scheduler outperformed the baseline almost often, even with retraining! The Raspberry Pi unit underperformed due to its hardware's architecture. The experiments showed that the dynamic user-level scheduler improves the system's throughput, which reduces attack detection time, and dynamically allocates resources based on attack suspicion. The results show that the suggested technique improves IDS performance for lightweight and data-driven ML algorithms for IoT.

### 2.3. Container Orchestration Applied to IoT Devices

Google's Borg was turned into Kubernetes using Go programming for Docker and Docker containers for orchestration. Kubernetes expanded on Google's Borg, which inspired its development. Later, Kubernetes was viewed as a strong edge computing solution, which led to the creation of lightweight container orchestration solutions like k3s, Microk8s, and others because Kubernetes required intensive hardware specs that IoT platforms lack. Performance of lightweight container orchestration technologies has been studied. All research is from the last two to three years.

The researchers [14] established a Kubernetes cluster using readily available Raspberry Pi devices. The cluster's capacity to handle IoT components as gateways for sensors and actuators was tested utilizing synchronous and asynchronous connection situations. Linux containers like Docker can be used to quickly deploy microservices on near-end and far-end devices. Container technologies isolate processes and hardware within an operating system. Due to its fast deployment, communication management, high availability, and controlled updates, Kubernetes may be a solution for IoT. The paper draws attention to the paucity of research on deploying Kubernetes in the IoT setting and the need for additional research to analyze its applicability and limitations. Performance testing, evaluating response times, request rates, and cluster stability, revealed the architecture's pros and cons [15].

Lightweight solutions have been developed in recent years. Minikube, Microk8s, k3s, k0s, KubeEdge, and Microshift. In several tests, the authors tested tiny clusters of lightweight k8s distributions under high workloads. These insights can assist researchers locate the lightweight k8s distribution for their use case. The researchers have basically found the resource consumption on idle for different k8s distributions, finding the resource consumption when the pods are being created, deleted, updated or being read and when the pods are assigned intensive workloads. Azure VMs, not IoT devices, powered the research. It would have been ideal to collaborate with IoT devices near to production. The researchers acknowledge the above shortcoming and suggest replacing artificial benchmark situations with production workloads and using black-box measures instead of white-box techniques. If the controller node has at least 1-2 GB of RAM and worker nodes have even



less hardware, all lightweight k8s distributions perform well on low-end single-board computers, according to the study.

#### *2.4. Mitigation to DDoS Attacks on IoT Devices*

While the detection work has been seen to be a work in progress, mitigation mechanisms after a DDoS attack happen are also being given due attention. Blockchain with the help of smart contracts is one interesting way of implementing this. Although, the solution at large is not localised to the IoT devices but seems to be making the solution work through the cloud.

Hayat et al. [16] explores IoT DDoS defenses, including smart contracts and blockchain-based methods. Multiplelevel DDoS (ML-DDoS) is being developed to protect IoT devices and compute servers against DDoS attacks using blockchain. It prevents devices turning into bots and increases security with gas restriction blockchain, device blacklisting, and authentication. Blockchain technology is also called a public ledger with unchangeable records. It facilitates peer agreements in asset management, finance, and other fields via consensus methods. The decentralized and open-source Ethereum blockchain is promoted for protocol-based transactional protocols between parties. comparison of ML-DDoS performance in the presence of bot-based scenarios shows that it outperforms other cutting-edge methods including PUF, IoT-DDoS, IoT-botnet, collaborative-DDoS, and deep learning-DDoS in terms of throughput, latency, and CPU use.

#### *2.5. Research Gap*

Table 1 highlights the issues of cyber security in IoT. Hardware complexity, linear cloud complexity increase that doesn't match IoT's exponential development, data manipulation by attackers, infrastructure botnets, and DDoS or Man in the Middle attacks on servers were mentioned. In the literature, deploying detection methods on routers for edge computing still leaves devices vulnerable because they are not secured. Additionally, infected devices are blocked from accessing the main servers. The detection model deployment should treat the device instead of rendering it useless. An IoT device with AI capabilities, including hardware configuration for machine learning algorithms, was proposed in [6]. In [13], Nvidia Jetson Nano, an AI IoT Device, was shown to have significant potential above Raspberry Pi. Even IoT research has advanced with container orchestration which is shown in [14,15]. These studies suggest that edge computing, container orchestration, AI IoT devices, and IoT devices in general can be combined to achieve more. This study combines various concepts to work cohesively.

Table 1. Literature Review

Ref	Work Done	Challenges
[6]	Attribution analysis of IoT Security Threats Security Threats in different layers of OSI model Discussion of Feasibility Analysis of AI in IoT Security AI solution for IoT security threats	Linear Growth of Cloud Computing does not match the exponential growth of IoT Devices. Computational Complexity for IoT Devices Data Manipulation done by at- tackers
[11]	a two-stage DDoS detection solution that utilizes deep learning algorithms and operates on the provider edge (PE) and consumer edge (CE).	Solution is not based on IoT devices but rather on routers.
[17]	ShadowNet deployed at the network’s edge to speed up defense against IoT-DDoS attacks.	Solution de- ployed on gate- ways
[14]	Exploring different Kubernetes distributions on rasp- berry pi clusters. Exploring pros and cons	Did not perform a load test on the cluster
[18]	Distributed platform over IoT devices so as to make them support GAN as a solution to Intrusion. Reduced dependency on Central cloud systems	The research didn’t present the CPU utilisation and Memory utilisation on the devices.
[15]	Analysis of different lightweight k8s distributions	Implemented it on Azure VMs.
[16]	Ethereum based blockchain Signature based authenticity test of devices	Eliminates the affected botnet devices
[12]	Kubernetes deployed on Raspberry Pi. Test conducted using synchronous and asynchronous scenarios to handle IoT components	The test could have also involved testing on machine learning paradigm as well.

3. Methodology

The CICDDoSDataSet2019 has been used for training the machine learning model. The dataset is shown in Table 2. This dataset has 2 Million entries and 80 columns after preprocessing. The preprocessing involved replacing spaces with no spaces in every column where ever applicable, dropping columns such as FlowID, SourceIP, DestinationIP, Timestamp, SimillarHTTP, SourcePort and DestinationPort. Since the numeric data was in text format so those were converted into the numeric format. There are 2180000 samples for DDoS packets and 1542 samples are benign samples. This indicates a heavy imbalance between the classes [19].

**Table 2.** DDoS Dataset

S.No.	Flow Duration	TotalFwd Packet	..	Similar HTTP	Inbound	Label
1	9141643	85894	..	0	1	DrDoS_LDAP
2	1	2	..	0	1	DrDoS_LDAP
3	2	2	..	0	1	DrDoS_LDAP
4	1	2	..	0	1	DrDoS_LDAP
5	2	2	..	0	1	DrDoS_LDAP
6	2	2	..	0	1	DrDoS_LDAP
7	2	2	..	0	1	DrDoS_LDAP
..	..	..	..	..	..	..
2181536	1	2	..	0	1	DrDoS_LDAP
2181537	50	2	..	0	1	DrDoS_LDAP
2181538	1	2	..	0	1	DrDoS_LDAP
2181539	1	2	..	0	1	DrDoS_LDAP
2181540	1	2	..	0	1	DrDoS_LDAP
2181541	1	2	..	0	1	DrDoS_LDAP
2181542	2	2	..	0	1	DrDoS_LDAP

### 3.1. DDoS Detection Model

Three Machine learning models were trained for the detection of DDoS.

1. **Dummy Classifier:** A dummy classifier is a simple and baseline machine learning model used for comparison and benchmarking purposes. It is typically employed when dealing with imbalanced datasets or as a reference to assess the performance of more sophisticated models. The dummy classifier makes predictions based on predefined rules and does not learn from the data. The dummy classifier with the strategy of predicting the most frequent class is utilised for the baseline. The most frequent class classifier always predicts the class that appears most frequently in the training data. This is useful when dealing with imbalanced datasets where one class is significantly more prevalent than others [20].
2. **Logistic Regression:** Logistic regression is a popular and widely used classification algorithm in machine learning. Despite its name, it is used for binary classification tasks, where the output variable has two classes (e.g., "Yes" or "No", "True" or "False"). The logistic regression model calculates the probability that an input data point belongs to a particular class. It does this by transforming its output through the logistic function (also known as the sigmoid function). The logistic function maps any real-valued number to a value between 0 and 1, which can be interpreted as a probability [21].
3. **Deep Learning Model:** The deep learning model has 3 layers to it. The first layer is the input layer. It has got the input shape of 80 neurons which outputs a shape of (None, 128). A dropout regularisation enables this layer to handle overfitting of the data. This is followed by another dense layer which also has a dropout regularisation. In the end, the model has an output layer with one neuron giving a probability as an output due to the sigmoid function being used in the last layer.

#### 3.1.1. Performance Metrics

1. **F1 score:** It combines the precision and recall scores of a model. It is usually more useful than accuracy, especially if you have an uneven class distribution. The equation is provided in equation 1.

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (1)$$

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \quad (2)$$



$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

(3)

3.2. Technology Used

3.2.1. Python

Python Programming Language has been used throughout the complete research. There are some important frameworks used as part of this research:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	10240
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65

Total params: 18,561  
Trainable params: 18,561  
Non-trainable params: 0

Figure 2. Model Definition of Keras Model

1.

Flask: This is a framework defined under Python language for hosting REST APIs. The programmer can define the business logic which interacts with the front end. It provides an interface between the end user and databases.
2.

Tensorflow (TF): This is a package built by Google for deep learning-based workloads. This package comes along with Keras. Keras is an Application Programming Interface which allows the programmer to build layers for Artificial Neural Networks. [22]

3.2.2. Microk8s

Microk8s has been developed by canonical and is being provided as a snap package as part of Ubuntu. It controls containerized services at the edge [23]. The previous researches have mentioned that the Microk8s requires atleast 4GB of RAM usage. The Microk8s initially ships with the basics of k8s for example the controller, API server, dqlite storage, scheduler and so on. Additionally, there are more add-ons that can be enabled by running “microk8s enable service name”. The name of the service can be Prometheus, DNS, metallb, helm, metrics-server, Kubernetes-dashboard and so on [15].

3.3. Components

This study used four Raspberry Pis with 4GB RAM each and one Nvidia Jetson Nano with 4GB RAM as part of the device infrastructure. Four MicroSD cards with 16GB capacity were used for Raspberry Pi and a 64GB MicroSD card was used for Nvidia Jetson Nano. Ubuntu Server 18.04 was installed on the Raspberry and Nvidia jetpack on the Nvidia Jetson Nano. The devices were connected with LAN wires and a switch as a medium between them. A WiFi adapter was also used with Nvidia Jetson Nano because Jetson doesn’t come with an onboard WiFi module. The switch model is Netgear JGS524E ProSafe Plus 24-Port Gigabit Ethernet Switch.

### 3.3.1. Nvidia Jetson Nano

A robust single-board computer made specifically for AI and robotics applications is the Nvidia Jetson Nano. The Nvidia Tegra X1 processor, which houses a quad-core ARM Cortex-A57 CPU and a 128-core Nvidia Maxwell GPU, powers the Jetson Nano [24]. This GPU is especially well-suited for jobs requiring parallel processing, making it the best choice for effectively executing deep learning models and AI workloads. There are other variations of the Jetson Nano available, but the most typical model comes with 4GB of LPDDR4 RAM, a microSD card slot for storage, Gigabit Ethernet, and USB 3.0 ports for fast data transfer. Its capabilities for robotics and computer vision are increased with the addition of GPIO ports, a MIPI CSI camera connection, and Display Serial Interface (DSI) for interacting with cameras and displays. The device is displayed in Figure 3.



Figure 3. Nvidia Jetson Nano

### 3.3.2. Raspberry Pi

The fourth version of the successful and adaptable single-board computer created by the Raspberry Pi Foundation is known as the Raspberry Pi 4. It became available in June 2019. It has a quad-core ARM Cortex-A72 CPU that runs up to 1.5GHz, giving it a substantial performance advantage over earlier iterations. The RAM is 4GB for the device being used in this project. The device is displayed in Figure 4 [25].



Figure 4. Raspberry Pi

3.3.3. Netgear Switch

Netgear JGS524E ProSafe Plus 24-Port Gigabit Ethernet Switch has got 24 ports on it. A gigabit connection delivers up to 2000 Mbps of dedicated, non-blocking bandwidth per port. It is a plug-and-play type of device. The device is shown in Figure 5.



Figure 5. Netgear Switch

3.4. Networking Architectures

The architecture figures that you would see below have a cloud named Network Cloud. The network cloud in general means any sort of networking interface can be used which could be a switch, WiFi network or maybe a private 5G network. For this study, a switch and WiFi have been used to network the device together.

3.4.1. Traditional Architecture and Nvidia Architecture

Figure 6a shows an architecture which has been used widely. This kind of architecture is used in a fashion to interact with the real world and collects data from the sensors connected to the Raspberry Pi. The devices like Raspberry Pi transport the data to the cloud for processing and running detection models. When this architecture is used, the computational workloads become a problem.

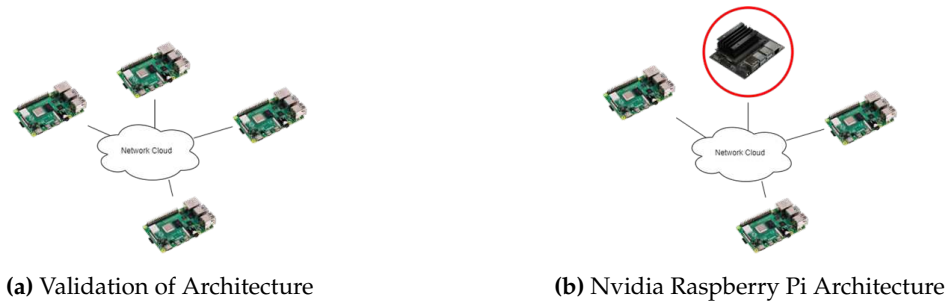


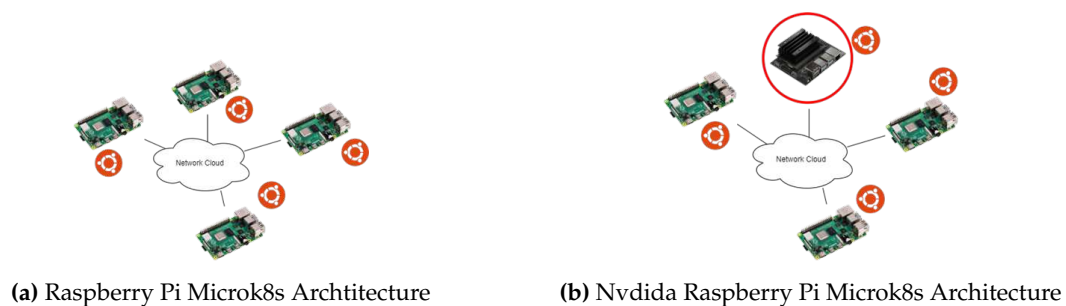
Figure 6. Traditional Architecture & Nvidia Architecture

As introduced previously in the general architecture utilise the AI-enabled IoT Device and replace one of the nodes in the Raspberry Pi architecture with AI-IoT Device. In this situation, the architecture is introduced with Nvidia Jetson Nano which is responsible for running various types of security breach detection models and transporting anomaly data to the cloud for further inspection. This way the amount of data stored in the cloud decreases by a major margin and goes light on storage as well. So, the only data which is stored are the anomalies which allow better analysis and deploy recovery automation solutions. The architecture is shown in Figure 6b.

3.4.2. Raspberry Pi Microk8s Archtitecture and Nvdida Raspberry Pi Microk8s Architecture

The architecture introduces Microk8s as a resource management and container orchestration tool. This architecture with Microk8s has been previously seen in various research. Although, these researchers have not performed major load testing on the cluster for example running a machine

learning model on the architecture. This study finds the CPU utilisation in the case of load testing related to running machine learning. The architecture is shown in Figure 7a.



**Figure 7.** Raspberry Pi Microk8s Architecture & Nvidia Raspberry Pi Microk8s Architecture

Nvidia Raspberry Pi Microk8s Architecture is introduced as the state-of-the-art architecture which includes the capabilities provided by GPU provided in Nvidia Jetson Nano as well as resource management tools like Microk8s. These both together provide exceptional abilities of efficient scheduling of memory and keeping track of microservices on the network as well as running the detection models with comparatively less CPU utilisation as compared to other network elements. The architecture is shown in 7b.

### 3.5. Validation of Hypothesis

#### 3.5.1. Machine Learning Validation

When it comes to Machine Learning model performance, it is expected that the model is able to predict classes distinctively without any major misclassification issue. In such scenarios, the F1 score associated with the classes should be substantial enough to make a feasible prediction to not fail when running on production systems. As we have seen earlier the F1 score is the weighted harmonic mean of Precision and Recall, the changes in the F1 score indicate an overall improvement as compared to the previous model. The Dummy classifier indicates the baseline performance that needs to be outperformed by other models. This study explores the logistic regression model and the neural network for the detection models. These models would be expected to perform better than the baseline and state-of-the-art neural network to perform better than Logistic Regression.

#### 3.5.2. Architecture Improvement Validation

For validation, it is expected that the traditional architecture will perform the poorest in terms of CPU utilisation and memory because of the hardware specifications of Raspberry Pi being used in the architecture. Therefore, the performance obtained in this architecture becomes the baseline which needs to be beaten by other architectures. Next, The Nvidia-enabled architecture is expected to utilise 20% less CPU as compared to traditional architecture. The Raspberry Pi architecture with Microk8s is expected to perform 10% less than the Nvidia enabled architecture because it is running an efficient container orchestration tool. In the end, the state-of-the-art architecture where the Nvidia device and Microk8s are together is expected to perform at least 60% less than the traditional architecture. This is demonstrated in Figure 8.

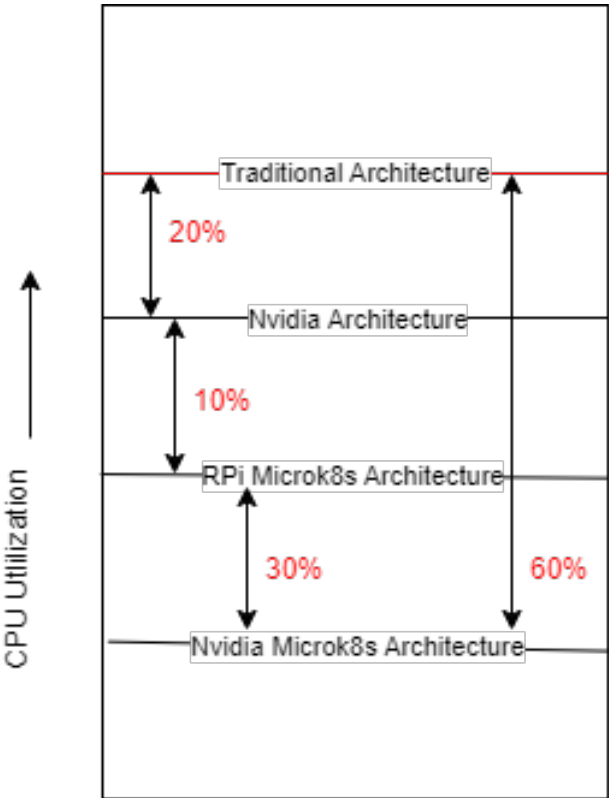


Figure 8. Validation of Architecture

4. Design and Implementation

4.1. Cluster Setup

Two network structures were used for this research:

4.1.1. Raspberry Pi Cluster and Nvidia-Raspberry Pi Cluster

In the cluster shown in figure 9a, four Raspberry Pis were used and one of them was made the master to handle operations in the Kubernetes environment. In a non-Kubernetes environment, the device was used to host the message queue Flask server and run the machine learning model.



(a) Raspberry Pi Cluster



(b) Nvidia-Raspberry Pi Cluster

Figure 9. Raspberry Pi Cluster & Nvidia-Raspberry Pi Cluster

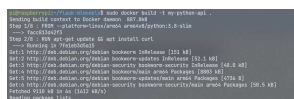
In the cluster shown in figure 9b, three Raspberry Pis and one Nvidia Jetson Nano were used and one of the Raspberry Pis was made the master to handle operations in the Kubernetes environment. In a non-Kubernetes environment, the Nvidia device was used to host the message queue Flask server and run the machine learning model.

#### 4.2. Docker

Docker [26] is a containerisation service which is used to build microservices. For all the microservices i.e. the Flask server for the message queue, the ML model and the data sending micro-services were converted into respective Docker images. These all microservices have Docker files in their directory. These Docker files can build using "docker build -t < name of the build > .". Once the build is complete, the Docker image needs to be pushed to the Docker hub. An account on the Docker hub is required and 3 Docker hub repositories are required. For every repository, the Docker image build previously needs to be pushed. For doing so, first the image needs to be tagged against the path of the repository location on the Docker hub. For example, "docker tag < name of user > / < name of online repository > : < tag > < name of the build >". Once this is done, the "docker push < name of user > / < name of online repository > : < tag >" command needs to be executed.

#### 4.3. Docker Implementation

In this section, an example of building and pushing a Docker image of a Machine learning microservice to the Docker hub is shown. In Figure 10a, the command executes to build the image for the machine learning model. Before pushing the Docker image to the Docker hub, it's required to first authorize your push by login into your ID associated with the Docker repository. Figure 10b displays an example of Docker login and success at the end of the image. One last step before pushing the image is to associate the name of the Docker hub repository with the name of the local Docker build. This is done by the "docker tag" command used in Figure 11.



```

pi@raspberrypi:~/python-api$ docker build -t my-python-api .
Sending build context to Docker daemon  88.46kB
Step 1/10 : FROM python:3.9-slim
Step 2/10 : RUN apt-get update && apt-get install -y python3-pip
Step 3/10 : COPY requirements.txt /
Step 4/10 : RUN pip install -r requirements.txt
Step 5/10 : COPY . /app
Step 6/10 : WORKDIR /app
Step 7/10 : CMD ["python", "main.py"]
Step 8/10 : EXPOSE 8080
Step 9/10 : HEALTHCHECK --interval=30s --timeout=30s --start-period=30s --retries=3 CMD curl -f http://localhost:8080/ || exit 1
Step 10/10 : COMMIT my-python-api
Successfully built my-python-api

```

(a) Docker Build



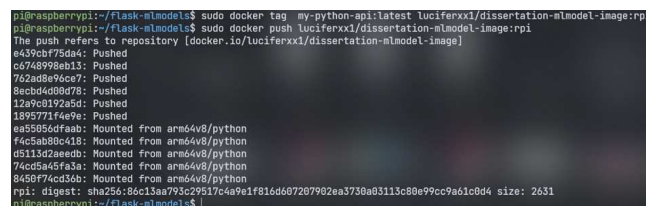
```

pi@raspberrypi:~/python-api$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one
Username: luciferxx1
Password:
Login Succeeded

```

(b) Docker login

Figure 10. Docker Login



```

pi@raspberrypi:~/python-api$ docker tag my-python-api:latest luciferxx1/dissertation-mlmodel-image:pi
pi@raspberrypi:~/python-api$ docker push luciferxx1/dissertation-mlmodel-image:pi
The push refers to repository [docker.io/luciferxx1/dissertation-mlmodel-image]
e439cbf75da4: Pushed
c6748998eb13: Pushed
7c2a8d989a97: Pushed
8ecbd4d8d78: Pushed
12a9c0192a5d: Pushed
1895771f4e9e: Pushed
e2585c5df8ab: Mounted from arm64v8/python
f4c5ab80c415: Mounted from arm64v8/python
d5113d2aeed0: Mounted from arm64v8/python
74cd5a45fa3a: Mounted from arm64v8/python
8458f74cd36b: Mounted from arm64v8/python
pi$ digest: sha256:86c13aa793c9517c4a9e1f816d607287902ea3738a0311c88e99cc9a61c8d4 size: 2631
pi@raspberrypi:~/python-api$

```

Figure 11. Docker Tag and Push

#### 4.4. Kubernetes Terminologies

Kubernetes is an open-source container orchestration platform used for automating the deployment, scaling and management of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes provides a powerful and flexible framework for managing containerized applications across a cluster of machines [27].

##### 4.4.1. Pods

The smallest deployable compute units that Kubernetes allows you to construct and control are called pods. A collection of one or more containers, with common storage and network resources, and a specification for how to execute the containers, is referred to as a "pod" (as in a pod of whales or peas).



The components of a pod are always co-located, co-scheduled, and executed in the same environment. A pod includes one or more closely connected application containers and serves as a representation of an application-specific "logical host". Apps running on the same physical or virtual system are comparable to cloud apps running on the same logical host in non-cloud scenarios [28].

#### 4.4.2. Deployment

A deployment is a crucial object in Kubernetes that controls the scaling and deployment of a collection of identical pods. One of the higher-level abstractions offered by Kubernetes to make managing containerized apps easier is this one. Using a deployment, you may provide the container image, the number of replicas (identical pods), and the update method to describe the intended state of your application [29].

#### 4.4.3. Namespaces

A method for separating groupings of resources inside the same cluster is provided by namespaces. Within a namespace, but not between namespaces, resource names must be distinctive. Only namespace-based scoping is applicable to cluster-wide items, such as Storage Class, Nodes, and Persistent Volumes, not namespace-based objects (such as Deployments, Services, etc.) [30].

#### 4.5. Design Flow

A visual representation of the workflow of the initial design is shown in Figure 12. The idea initially for the workflow about traditional implementation was to implement a complete DDoS happening environment where a real test on the devices could be done. The design goes as the record of performance metrics would be commenced to be written into a file followed by starting the Flask server on the master node. Once these steps are done, the worker nodes would be administered a SYN Flood attack from the Kali virtual machine running on the laptop. Simultaneously, a Docker container of cicflowmeter would be started.

CICFlowmeter is a software which allows the network-related data stored in a PCAP format to be transformed into the CSV format of the network logs based on the physical interfaces of the device. This CSV data would be sent to the Message queue running on the Flask server. Once these messages are received, the Machine learning script would be executed in a cronjob timed 1 minute which would first check for availability of the logs, if there are any, it will start fetching until the message queue is empty. This keeps on looping until the Flask server is stopped or crashes due to some reason. Once the Flask server is stopped, the logs can be stopped recording and stored.

The design described above could not be followed because CICFlowmeter is an archive library which has outdated dependencies. Even though those dependencies were configured, the program was not able to function as needed. The other option available was to use a Docker container but that didn't work as well since the container would get crashed due to Java dependencies issues. So, the only way to use to CICFlowmeter was to directly generate a PCAP file on a Windows computer and then utilise the CSV file generated which was not feasible in this research's use case because the cicflowmeter workflow needs to be automated.

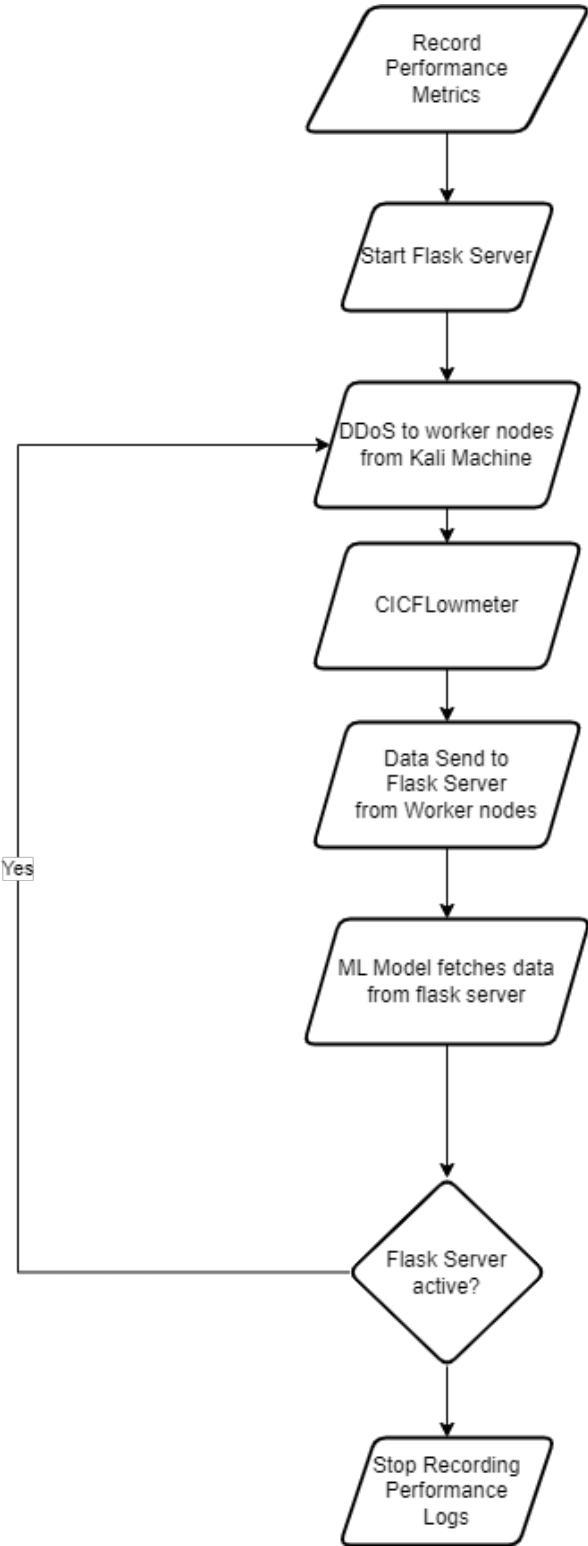
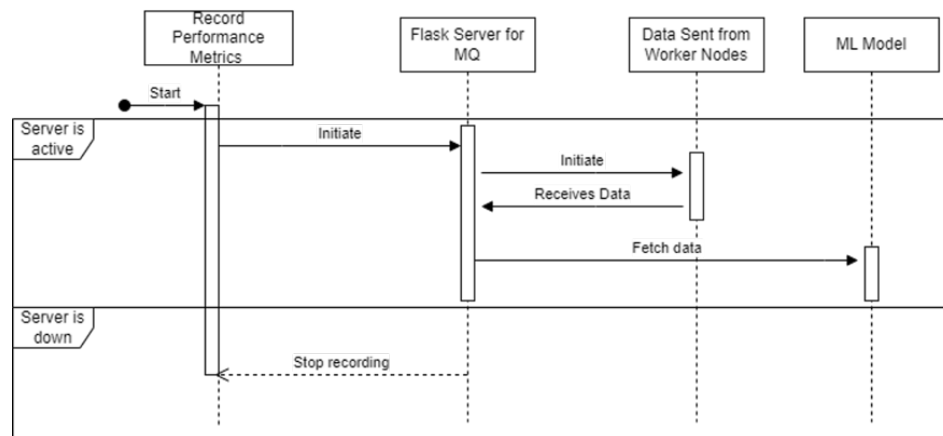


Figure 12. Initial Design Flow

4.5.1. Message Queue Implementation

The intuition behind having a message queue is to store the logs for the machine learning model to fetch from. For a message queue (MQ), a First In First Out(FIFO) implementation is required. A normal queue has a fixed size for the array. But for this use case of message queue, the dynamic length of an array is required which can extend based on logs being inserted from worker nodes. Therefore, Doubly

Linked List becomes an eventually important choice as the list can be traversed bidirectionally and insertions and deletions can be done from both the end. Due to initial design issues, the CICFlowmeter was skipped. The data set provided above sends data to the message queue. The message queue Flask server is executed after hardware performance metrics are recorded. After the Flask server gets requests, the script to send data is executed, and the message queue stores the data until the ML model script demands it. The ML model script checks if the message queue is active and then processes data from it. After using message queue data, the ML model scripts stop and run on a 1-minute cronjob time. The above is presented in a visual form in the figure 13.



**Figure 13.** Work Flow without Kubernetes

For the Kubernetes cluster, different devices need to be first formed into a cluster. For that, one of the nodes is decided as a master node. The /etc/hosts file is updated with the hostname of the other devices and their respective static IP addresses. It is necessary that all the devices are connected to the internet and same access point. Once this is done, the "microk8s add-node" run on the master node, which provides a connection string which needs to be executed on the other devices so that they know which is the master node. After this, additional services need to be enabled on the cluster. The services are as follows:

1. DNS (Domain name service): This is required for the pods to resolve the flask-app- service internally when the machine learning/data-sending script is executed.
2. Metallb ( Metal Load Balancer): This load balancer is required to load balance if there are multiple pods for flask service running.
3. Ingress
4. GPU: This only applies to Nvidia-based hardware and given that the device should have an Nvidia GPU. The Microk8s detect the GPU and execute the tasks which require GPU if the deployment is launched.
5. Metrics-server: This is required for recording the performance metrics of the Kubernetes cluster. This service enquires about the utilisation of CPU and memory for every pod running in a particular namespace.

After finishing the instructions, start the pods. Before Flask deployment, the performance metrics recording script is run. Flask-app-service must be checked after starting the flask pod. To verify the status of the flask-app-service, use "kubectl get services". Data-send deployment can begin after the pod is up. When data-send pods are active, Flask receives data via internal domain name resolution. To verify requests, use "kubectl logs < name of flask pod >". The name of the Flask pod may be obtained with "kubectl get pods | grep flask". After seeing the requests, the deployment file's cronjob option can perform and clock the machine learning model deployment every minute. Requests appear in Flask server logs after execution. This completes the flow shown in figure 14. The monitor script can be stopped and the performance metrics can be noted.

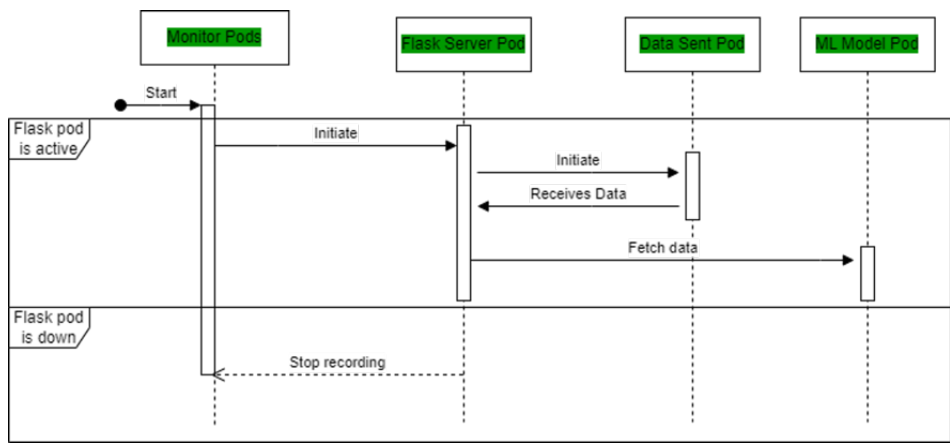


Figure 14. Kubernetes Work Flow Diagram

4.6. Implementation of Non-Microk8s Architecture

4.6.1. Performance Metrics

In Figure 15, it shows the execution of the command to start recording the performance parameters of CPU and Memory. For this, 'while true' is used to execute a never-ending loop until the script is stopped. echo "%CPU %MEM ARGS \$(date)" is printed at the top of every output which is printed into the file.

```
pi@raspberrypi:~/logs$ echo "This is the terminal for Monitoring the CPU And Memory utilization"
This is the terminal for Monitoring the CPU And Memory utilization
pi@raspberrypi:~/logs$ while true; do (echo "%CPU %MEM ARGS $(date)" && ps -e -o pcpu,pmem,args --sort=pcpu | cut -d" " -f1-5 | tail) >> ps.log; sleep 5; done
```

Figure 15. Performance Metrics Collection

The provided Linux shell command is a pipeline that extracts and logs information about processes on the system with CPU usage. It begins by executing the ps command with the -e flag, which selects all processes running on the system. The -o pcpu,pmem,args flag customizes the output format, displaying CPU usage percentage, memory usage percentage, and the command with its arguments. The results are then sorted based on CPU usage using the -sort=pcpu flag. The cut command is used with the -d flag to extract the first five columns (CPU usage, memory usage, and command) from each line. The 'tail' command displays the last few lines of the sorted output, representing processes with the highest CPU usage. Finally, the » ps.log command appends the selected output to a file named "ps.log", providing users with a log of processes' CPU usage for analysis and future reference. And sleep 5 is used for recording entries every 5 seconds. This script can be stopped once the Flask server is killed or stopped.

4.6.2. Flask Execution & Data Sent Execution

Figure 16a displays the Rest API server deployment known as the Flask framework. Figure 16b displays the transfer of information to the REST API server. The "Success" print presents the successful request to the REST API server.



(a) Flask Execution

(b) Data Sent Execution

Figure 16. Execution

4.6.3. Machine Learning Execution

Figure 17a displays the execution of the Machine Learning model on the Raspberry Pi platform. At the end of the verbose, the prediction is displayed. The implementation of the Nvidia enabled architecture remains the same as mentioned for the Raspberry Pi. The machine learning model and the Flask script runs on the Nvidia device. In Figure 17b, the Nvidia Tegra X1 is detected as GPU because Tegra X1 contains the CPU and GPU inside it. The CPU's clock speed is displayed as .91GHz. The memory of the device is displayed as 3.86 GB.



(a) Machine Learning Script Execution (b) Nvidia Machine Learning Script Execution

Figure 17. Script Execution

In Figure 18, 'cat' is used for reading the file of performance metrics. The output of this command is piped together with a word selection command called "grep". Grep is used to select those lines where the word occurs. For example, it shows the line where the word dl\_new.py occurs.

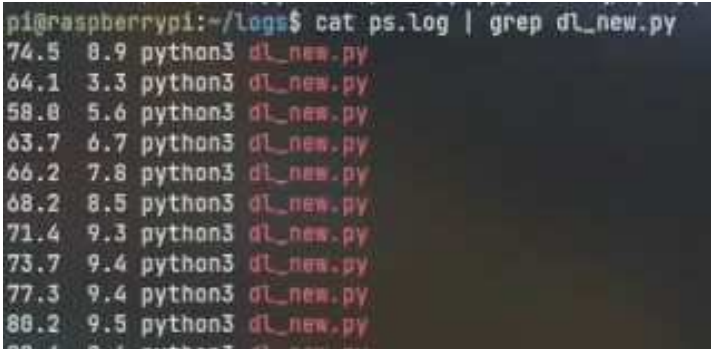


Figure 18. Viewing the Performance Metrics

4.6.4. Reading Performance File

4.7. Implementation of Microk8s Architecture

In Figure 19a, the nodes running in a cluster are enlisted. The image displays all the Raspberry Pi online at the moment when the command "kubectl get no" was executed. Figure 19b presents the execution of the deployment file called "flask-app-dep.yml".



(a) "kubectl get no" command (b) Flask App Deployment

Figure 19. Flask Deployment

4.7.1. Performance Metrics

For performance metrics to be recorded, the metrics-server add-on needs to be enabled on Microk8s. With this enabled, a shell script is used to get the output of the command "microk8s kubectl top pods" to receive the information of CPU and Memory utilisation into a file. This information gets recorded every 10 seconds while the script is running. Figure 20a displays the "kubectl describe po" command being executed while figure 20b shows the output of the command. In the output, the pulling image

signifies the cluster is communicating Docker hub to fetch the Docker image while the statement above it resembles the successful image pull and the master node assigning the job to a node automatically if not specified otherwise.

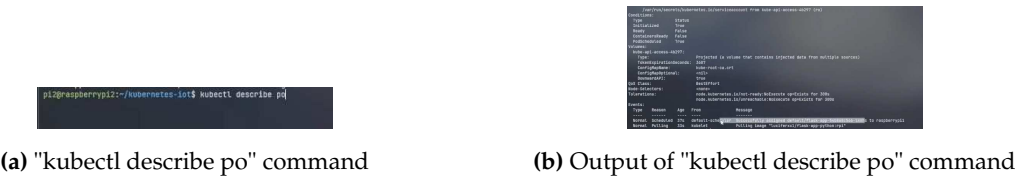


Figure 20. "Kubectl describe po" command

Figure 21a displays the output of the command "kubectl get pods". This command is followed by a "-o" flag with the keyword "wide" which displays additional information such as the node, IP address, Nominated Node and Readiness Gate. The image signifies the Flask pod in the creation stage and the pod is running on the raspberrypi1 node.

Figure 21b displays the IP addresses on which the Flask app can be accessed. Kubernetes allows domain name resolution which could be used by other pods to resolve "flask-app-service" into the corresponding internal cluster IP addresses.



Figure 21. "Kubectl get" command

Therefore, no hard coding is required for the IP addresses in the scripts of the data-sending and machine-learning model. The external IP address is the IP address which is not the IP address of any node inside the cluster but a different address altogether which allows the "flask-service" to be accessed from the external world. Since this address does not correspond to any node, it provides extensive security to the nodes as they are hidden from the external world and thus remain protected from attacks. Also, the external IP address is provided by a load balancer called "metallb". This load balances the requests across the pods running the Flask app deployment.

Figure 22a displays the container logs of the pod running the Flask API microservice. It shows the Flask server is up and running. The 127.0.0.1:5001 is the local host API deployment which can be accessed from the pod itself while the 10.1.245.3:5001 is the IP address for the internal resolution to the other pods for accessing the service if required. Figure 22b shows the execution of the "csv-data-dep.yml". This YML file holds the name of the Docker image which needs to be sourced from the Docker hub. This deployment is responsible for sending the information to the Flask deployment.



Figure 22. Deployment

Figure 23a shows the logs of the Flask pod after the csv data-sending pods use the Flask server's endpoint to send the data. Once the request is served by the Flask, it logs into the output shown. Figure 23b shows the execution of the "ml-model-dep.yml". This YML file holds the name of the Docker image which needs to be sourced from the Docker hub. This deployment is responsible for retrieving the data from the Flask server for running the machine learning model.





4. **Prometheus and Grafana** These services are offered as add-ons to the Microk8s (version 1.22+). They are responsible to capture resource utilisation of every pod in each namespace or node in the cluster. Prometheus and Grafana were initially tested on the cluster. Still, the resource utilisation by the Prometheus namespace was high which was crashing the Microk8s service on the nodes when the other deployments were being initiated. Therefore, it is suggested to have at least 8GB of RAM on the node’s hardware.

5. Evaluation

5.1. Machine Learning

The classes which are classified as 0 are the DDoS Packets while 1 indicates the normal packets.

Dummy Classifier Performance

In Figure 26a, The F1 score for class 0 (DDoS) is predicted as 1.00 because it has samples with the highest frequency. This is supported by the reason that the strategy for the dummy classifier was set as the most frequent. The F1 score of class 1 is 0 because it has fewer samples as compared to class 0.

Logistic Regression Performance

In Figure 26b, The F1 score for the benign class improves more than it was previously 0 in the baseline model. There is an improvement over there.



Figure 26. Dummy Classifier & Logistic Regression Performance

Neural Network Performance

In Neural Network, a regularized architecture has been defined. There is a 3% improvement as compared to the F1 score of class 1 in Logistic Regression Performance.

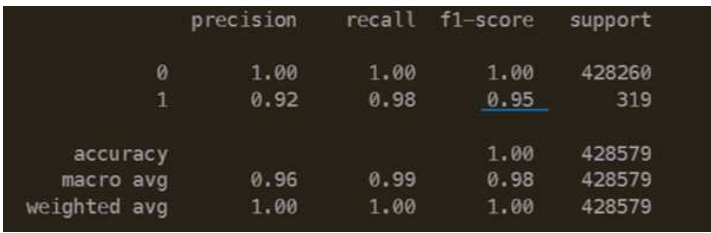


Figure 27. Neural Network Performance

Given all of this above architecture, the prediction for class 0 is an absolute 100 because of the biased samples in the dataset. Even if the undersampling is done, the relative samples for class 1 would still be undermined. Therefore, an improvement in such scenarios is difficult in this kind of unbalanced situation. Although, the model can be improved when the real data comes into the IoT devices by retraining the network in real-time.

5.2. Architecture

In Table 3, the CPU and memory utilisation is displayed for all the architectures obtained from the experiments. The data in the table is sourced from the experiments conducted and the evidence has been provided below in the figures. Looking at the table, it’s observable that the Non-Microk8s

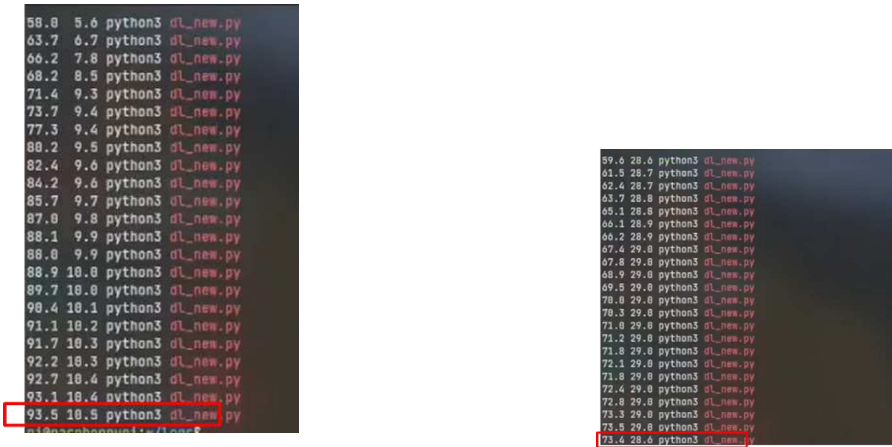
architecture has the highest CPU utilisation compared to the others. The CPU utilisation drops down to 73% when the Nvidia-enabled architecture is brought in. There is definitely an improvement over there which showcases one of the reasons where GPU assists the CPU in running the machine learning models in an efficient manner. While having looked at the memory utilisation it's increasing because Nvidia uses a Graphic-based User Interface of Ubuntu which also runs other System operations as well whereas the memory utilisation is less because of the server image of Ubuntu being used.

Discussing the Microk8s architectures, the Raspberry Pi architecture performs better in terms of both CPU and Memory utilisation as compared to non-microk8s architecture and interestingly 10% less than the Nvidia Non-Microk8s architecture. Microk8s does a good job of managing the resources. While looking at the State-of-the-Art architecture where the Nvidia and Microk8s have been used. A CPU utilisation of 31.5% is noticed which is 60% of the traditional architecture.

Table 3. CPU and Memory Usage Comparison

	Non-Microk8s (H5 Model)		Microk8s (TFLite Model)	
	Rpi (Traditional)	Nvidia-Rpi	Rpi	Nvidia-Rpi
CPU (%)	93.5	73.4	64.2	31.5
Memory (%)	10.5 (420 MB)	28.6 (1120 MB)	0.0007 (3 MB)	0.01 (73 MB)

Figure 28a, displays the CPU utilisation in the leftmost column while the column beside it displays the Memory utilisation of the Raspberry Pi traditional architecture. The 'dl\_new.py' is the file which holds the logic to fetch the data from the message queue and run the machine learning model once the data is available in the memory. The image, therefore, shows the continuous CPU consumed with time and a constant rate of increase.



(a) Raspberry Pi Traditional Architecture Performance (b) Nvidia Enabled Architecture Performance

Figure 28. Raspberry Pi Traditional & Nvidia Enabled Architecture Performance

Figure 28b presents the CPU utilisation in the leftmost column while the column beside it displays the Memory utilisation of the Nvidia-enabled architecture. The Nvidia-enabled architecture starts with a similar amount of CPU utilisation when the script initiates but over time the difference between traditional and this architecture grows larger. In the end, the final CPU utilisation is observed to be 73.4%.

Figure 29a & Figure 29b presents the name of the pods, the CPU utilisation and memory utilisation for the Raspberry Pi Microk8s architecture & Nvidia-Raspberry Pi Microk8s Architecture respectively. In the red box, the machine learning pod can be seen with the CPU utilisation and Memory utilisation beside it. The CPU utilisation is 64% & 31.5% respectively.



(a) Raspberry Pi Microk8s Performance (b) Nvidia Raspberry Pi Microk8s Performance

Figure 29. Raspberry Pi Microk8s & Nvidia Raspberry Pi Microk8s Performance

6. Conclusion

This research investigates distributed computing on the edge, using AI-enabled IoT devices and container orchestration tools to handle data in real time. Security is improved by identifying and mitigating threats like DDoS attacks while minimizing CPU usage. It compares typical IoT devices with and without AI-enabled chips, container orchestration, and machine learning model performance in different cluster settings. By enabling IoT devices to process data locally, the suggested design seeks to reduce reliance on cloud transmission and improve IoT environment security. Results align with architecture update. The following statements are concluded after looking at the results:

- 1. Comparison Of Raspberry Pi to Microk8s Raspberry Pi: Reduction by almost 30% (CPU) & 99% (Memory)
- 2. Comparison Of Nvidia - Raspberry Pi to Microk8s Nvidia - Raspberry Pi: Reduction by almost 42% (CPU) & 98% (Memory)
- 3. Comparison of Raspberry Pi and Nvidia-Raspberry Pi: Reduction by almost 21% (CPU) & Increase in 62.5% (Memory)
- 4. Comparison of Microk8s Architecture – Raspberry Pi vs Nvidia-Raspberry Pi: Reduction by almost 32.5% (CPU) & Increase in 95% (Memory)
- 5. An overall decrease of 60% in CPU utilisation from the traditional architecture of Rasp berry Pi to Microk8s architecture with Nvidia.
- 6. Container Orchestration-as-a-solution: It managed resources efficiently, auto-scaled when needed, and scheduled jobs well. Since the solutions are not directly distributed to the host, malware implantation can be stopped by restarting another pod for that deployment, making it security-reliable.

6.1. Future Work

- 1. Use of Google Coral Device: The Google gadget has a Tensor Processing Unit. Eventually, this gadget and Raspberry Pi can be compared to Nvidia Jetson Nano for performance analysis. This can be clustered or standalone.
- 2. Realtime updating of Machine Learning Model: When data becomes available on the platform, cloud updating the machine learning model affects model’s real-time performance reliability. If this could be done on the edge with CPU optimization in mind. Imagine what IoT devices could achieve with merely data coming in and the model being updated in real time without sending it to the cloud.
- 3. Adversarial AI for Malware Detection: Attackers must escape IoT devices after planting botnets to avoid leaving cyber fingerprints. Adversarial AI that predicts evasion and avoids malware plants. This would preserve the attackers’ fingerprints and make tracking malware to its source easier. Generative Adversarial Network implementation in malware detection is also receiving interest.

Looking at the above results and future work, there is tremendous potential which can be unlocked with the introduction of Kubernetes and AI-enabled IoT devices to the existing network of IoT devices. This will yield better product lineups for industries in this business of work. With the scale at which IoT devices grow, these solutions will be able to keep security growing as well.

**Author Contributions:** Conceptualization, S.K. and H.T.; methodology, S.K.; software, V.T.; validation, S.K., H.T. and V.T.; formal analysis, S.K.; investigation, H.T.; resources, S.K.; data curation, V.T.; writing—original draft preparation, S.K.; writing—review and editing, S.K. and H.T.; visualization, V.T.; supervision, H.T.; project administration, H.T.; funding acquisition, H.T. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was conducted with the financial support of Ripple.com under their University Blockchain Research Initiative (UBRI), and Science Foundation Ireland at ADAPT, the SFI Research Centre for AI-Driven Digital Content Technology at Trinity College Dublin [13/RC/2106\_P2].

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Merenda, M.; Porcaro, C.; Iero, D. Edge machine learning for ai-enabled iot devices: A review. *Sensors* **2020**, *20*, 2533.
2. Ghosh, A.; Chakraborty, D.; Law, A. Artificial intelligence in Internet of things. *CAAI Transactions on Intelligence Technology* **2018**, *3*, 208–218.
3. Covi, E.; Donati, E.; Liang, X.; Kappel, D.; Heidari, H.; Payvand, M.; Wang, W. Adaptive extreme edge computing for wearable devices. *Frontiers in Neuroscience* **2021**, *15*, 611300.
4. Fayos-Jordan, R.; Felici-Castell, S.; Segura-Garcia, J.; Lopez-Ballester, J.; Cobos, M. Performance comparison of container orchestration platforms with low cost devices in the fog, assisting Internet of Things applications. *Journal of Network and Computer Applications* **2020**, *169*, 102788.
5. Taylor, R.; Baron, D.; Schmidt, D. The world in 2025—predictions for the next ten years. In Proceedings of the 2015 10th International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT). IEEE, 2015, pp. 192–195.
6. Wu, H.; Han, H.; Wang, X.; Sun, S. Research on artificial intelligence enhancing internet of things security: A survey. *Ieee Access* **2020**, *8*, 153826–153848.
7. Shakhder, A.; Agrawal, S.; Yang, B. Security vulnerabilities in consumer iot applications. In Proceedings of the 2019 IEEE 5th Intl conference on big data security on cloud (BigDataSecurity), IEEE intl conference on high performance and smart computing, (HPSC) and IEEE intl conference on intelligent data and security (IDS). IEEE, 2019, pp. 1–6.
8. Dinculeană, D.; Cheng, X. Vulnerabilities and limitations of MQTT protocol used between IoT devices. *Applied Sciences* **2019**, *9*, 848.
9. Pokhrel, S.; Abbas, R.; Aryal, B. IoT security: botnet detection in IoT using machine learning. *arXiv preprint arXiv:2104.02231* **2021**.
10. Alrowaily, M.; Lu, Z. Secure edge computing in IoT systems: Review and case studies. In Proceedings of the 2018 IEEE/ACM symposium on edge computing (SEC). IEEE, 2018, pp. 440–444.
11. Alrowaily, M.; Lu, Z. Secure edge computing in IoT systems: Review and case studies. In Proceedings of the 2018 IEEE/ACM symposium on edge computing (SEC). IEEE, 2018, pp. 440–444.
12. Bhardwaj, K.; Miranda, J.C.; Gavrilovska, A. Towards {IoT-DDoS} Prevention Using Edge Computing. In Proceedings of the USENIX workshop on hot topics in edge computing (HotEdge 18), 2018.
13. Mirzai, A.; Coban, A.Z.; Almgren, M.; Aoudi, W.; Bertilsson, T. Scheduling to the Rescue; Improving ML-Based Intrusion Detection for IoT. In Proceedings of the Proceedings of the 16th European Workshop on System Security, 2023, pp. 44–50.
14. Beltrão, A.C.; de França, B.B.N.; Travassos, G.H. Performance Evaluation of Kubernetes as Deployment Platform for IoT Devices. In Proceedings of the Ibero-American Conference on Software Engineering, 2020.
15. Koziolk, H.; Eskandani, N. Lightweight Kubernetes Distributions: A Performance Comparison of MicroK8s, k3s, k0s, and Microshift. In Proceedings of the Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering, 2023, pp. 17–29.
16. Hayat, R.F.; Aurangzeb, S.; Aleem, M.; Srivastava, G.; Lin, J.C.W. ML-DDoS: A blockchain-based multilevel DDoS mitigation mechanism for IoT environments. *IEEE Transactions on Engineering Management* **2022**.
17. Todorov, M.H. Deploying Different Lightweight Kubernetes on Raspberry Pi Cluster. In Proceedings of the 2022 30th National Conference with International Participation (TELECOM). IEEE, 2022, pp. 1–4.



18. Ferdowsi, A.; Saad, W. Generative adversarial networks for distributed intrusion detection in the internet of things. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019, pp. 1–6.
19. Elsayed, M.S.; Le-Khac, N.A.; Dev, S.; Jurcut, A.D. Ddosnet: A deep-learning model for detecting network attacks. In Proceedings of the 2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM). IEEE, 2020, pp. 391–396.
20. Kannavara, R.; Gressel, G.; Fagbemi, D.; Chow, R. A Machine Learning Approach to SDL. In Proceedings of the 2017 IEEE Cybersecurity Development (SecDev). IEEE, 2017, pp. 10–15.
21. Bapat, R.; Mandya, A.; Liu, X.; Abraham, B.; Brown, D.E.; Kang, H.; Veeraraghavan, M. Identifying malicious botnet traffic using logistic regression. In Proceedings of the 2018 systems and information engineering design symposium (SIEDS). IEEE, 2018, pp. 266–271.
22. Ma, L.; Chai, Y.; Cui, L.; Ma, D.; Fu, Y.; Xiao, A. A deep learning-based DDoS detection framework for Internet of Things. In Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC). IEEE, 2020, pp. 1–6.
23. Debauche, O.; Mahmoudi, S.; Guttadauria, A. A new edge computing architecture for IoT and multimedia data management. *Information* **2022**, *13*, 89.
24. Süzen, A.A.; Duman, B.; Şen, B. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In Proceedings of the 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). IEEE, 2020, pp. 1–5.
25. Gizinski, T.; Cao, X. Design, Implementation and Performance of an Edge Computing Prototype Using Raspberry Pis. In Proceedings of the 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2022, pp. 0592–0601.
26. docker docs. docker overview. <https://docs.docker.com/get-started/overview/>,, 2023. Accessed: July 2023.
27. Redhat. "What is kubernetes?". <https://www.redhat.com/en/topics/containers/what-is-kubernetes>,, 2023. Accessed: July 2023.
28. Kubernetes. "Pods". <https://kubernetes.io/docs/concepts/workloads/pods/>,, 2023. Accessed: July 2023.
29. Kubernetes. "Deployment". <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>,, 2023. Accessed: July 2023.
30. Kubernetes. "Namespaces". <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>,, 2023. Accessed: July 2023.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.