An Intelligent Edge System for Face Mask Recognition Application

Tuan Le-Anh(✉), Bao Nguyen-Van, and Quan Le-Trung

UiTiOt Research Group, Department of Computer Networks, University of Information Technology, Vietnam National University – Ho Chi Minh City, Ho Chi Minh City, Vietnam
tuanla.14@grad.uit.edu.vn, {baonv,quanlt}@uit.edu.vn

Abstract. In the modern age, the growth of embedded devices, IoT (Internet of Things), 5G (Fifth Generation) and AI (Artificial Intelligence), has driven edge AI applications. Adopting Edge computing for AI applications intends to deal with power consumption, network capacity, response latency issues. In this paper, we introduce an intelligent edge system. It aims to assist with managing and developing microservices based AI applications on embedded computers with limited hardware resource. The proposed system uses Docker/Containerd and lightweight Kubernetes cluster (K3s) for high availability, self-healing, load balancing, scaling and automated deployment. It also facilitates GPU (Graphics Processing Unit) to speed up AI applications. The centralized cluster management and monitoring features simplify clusters and services administration, especially on a large scale. Meanwhile, container registry and DevOps platform with built-in code repository and CI/CD (Continuous Integration/Continuous Delivery) offer continuous integration and delivery for AI applications running on the cluster. This improves the process of AI applications development and management at the edge. In this experience, we implement the face mask recognition application with the proposed system. This application engages the state-of-the-art and lightweight object detection models with deep learning, observing mask violations to contribute to reducing the spread of COVID-19 disease.

Keywords: Edge computing · IoT · AI · Docker/Containerd · Kubernetes · DevOps · CI/CD · Cluster management · Monitoring

1 Introduction

In modern times, the figure of devices connected to the network, and the data generated by these devices in many fields have been increasing [1, 2], in which processing data and applying AI majorly occur in the cloud. However, the IoT-Cloud approach encounters with response latency, energy consumption, and network capacity issues. To work out the situation, the IoT-Edge-Cloud approach has been in place. This brings the computation to the edge, close to the IoT devices sending the data. Besides, AI at the edge has become one of the top research trends in recent years [3]. It has strengthened real-time AI applications with the back of modern embedded systems, IoT devices, and state-of-the-art deep learning models. By taking advantage of container-based virtualization

N.-S. Vo et al. (Eds.): INISCOM 2022, LNICST 444, pp. 107–124, 2022.
https://doi.org/10.1007/978-3-031-08878-0_8

and lightweight container orchestration technologies, AI applications run on isolated environments with high availability, self-healing, load balancing, scaling and automated deployment [4, 5]. In addition, applying cluster management and monitoring features makes it effortless to administer clusters and AI applications as services on the cluster. For systems on a large scale, these will be more beneficial. The container registry and DevOps framework with built in Git repository and CI/CD for continuous delivery of AI applications. This speeds up the development and management of AI applications.

In this study, we present an intelligent edge system to aid with managing and developing microservices based AI applications on embedded computers with hardware constraints. The proposed system engages the approaches mentioned above. In which, we apply the proposed system for face mask recognition application.

This paper follows this structure: Sect. 1 introduces the approaches relevant to the proposed system. Section 2 outlines related work. Section 3 presents the proposed system's design and implementation. Section 4 shows the experimental results. Section 5 ends with a conclusion and future work.

2 Related Work

This section presents an overview of research on smart COVID-19 pandemic observation systems, edge AI and containerization, and state-of-the-art object detection models for real-time AI applications at the edge.

2.1 COVID-19

The need to apply AI to deal with COVID-19 has been huge over the years. In which AI-based observation applications have been used to help mitigate the spread of the disease. These include social distancing, body temperature monitoring, and mask detection [6–8] in the area we cover in this article.

2.2 Edge Artificial Intelligence and Containerization

In recent years, Edge AI has been growing, especially for real-time AI applications with low latency, network capacity and energy saving. It favors container-based virtualization for flexible and fast deployment. Some major container-based technologies include Docker, Containerd, CRI-O with GPU support. These platforms run containers with a performance like a bare-metal environment. In addition, container orchestration platforms such as lightweight Kubernetes (K3s) also improve edge AI performance by optimizing workload placement. Nowadays, Kubernetes is one of the top choices for automating deployment, scaling, and management of containerized applications. Some leading managed service and certified Kubernetes include Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS), etc. For edge or on-premises systems, major distributions are MicroK8s, K3s, and so on. K3s is one of the best candidate for edge systems with lower resource usage than MicroK8s [9]. Some relevant case studies include A Container-Based Edge Computing System for Smart Healthcare Applications [5], Edge Computing and Artificial Intelligence for Landslides Monitoring [10], Edge AI-IoT Pivot Irrigation, Plant Diseases and Pests Identification [11].

2.3 Deep Learning-Based Object Detection

In computer vision, there are many studies about real-time object detection models with deep learning. In this section, we conducted a survey of Faster R-CNN, Mask R-CNN, SSD, and YOLO. In which lightweight YOLO models balancing between the performance and accuracy would be our choice to apply for facemask recognition on embedded systems, ARM architecture with limited hardware resources.

2.3.1 Faster R-CNN and Mask R-CNN

Shaoqing Ren et al. introduced Faster R-CNN in 2016 [12]. The model uses Region Proposal Network to predict Region of Interest. Faster R-CNN outperformed with 0.2 s per image, compared to 2.3 and 49 s of Fast R-CNN and R-CNN respectively, using VGG16 network. Kaiming He et al. introduced Mask R-CNN in 2018 [13] with adding a new branch to predict mask and the rest was the same as Faster R-CNN. Although introducing a minor computational cost to detect mask, Mask R-CNN still achieved of 5 FPS.

2.3.2 SSD

Wei Liu et al. proposed SSD (Single Shot MultiBox Detector) in 2016 [14]. The model uses a single stage to predict bounding boxes and class probabilities from the entire image. Meanwhile, Faster R-CNN runs two stages. SSD can achieve high accuracy with relatively low resolution input images, speeding up object detection. SSD300 and SSD512 got performance of 46 FPS and 19 FPS respectively, outperforming Faster R-CNN with 7 FPS. Meanwhile, their accuracy was of 74.3 mAP and 76.8 mAP respectively, higher than Faster R-CNN with 73.2 mAP. Compared to YOLOv1 using VGG-16, SSD300 was more than twice, up to 46 FPS compared to 21 FPS and their accuracy was also higher, reaching 74.3 mAP compared to YOLOv1's 66.4 mAP.

2.3.3 YOLO

Joseph Redmon et al. introduced YOLO (You Look Only Once) in 2016 [15]. Similar to SSD, YOLO is a one-stage object detection model. It outperformed Faster R-CNN, achieving 45 FPS, although its 63.4% mAP sacrificed a bit, compared to Faster R-CNN model. Besides, YOLO-Tiny achieved 155 FPS and 52.7% mAP on PASCAL VOC. Joseph Redmon et al. proposed YOLO9000 and YOLOv2 in 2016 [16]. YOLO9000 could detect 9,000 different objects. YOLOv2 got 76.8% mAP on VOC 2007 at 67 FPS on 2007 and 2012 VOC using Darknet-19 network and new enhancements. This result outperformed Faster R-CNN and SSD. Joseph Redmon et al. introduced YOLOv3 in 2018 [17] with Darknet-53 network. Its major enhancements include multi-label prediction, predictions across scales etc. YOLOv3-320 achieved 45.5 FPS and 51.5% mAP with IoU 0.5 on COCO trainval. YOLOv3-Tiny offered high performance at 220 FPS on COCO, although its 33.1% mAP sacrificed, compared to YOLOv3. Another research, Mini-YOLO [18] got 52.1% mAP at 67 FPS. Alexey Bochkovskiy et al. brought birth to YOLOv4 in 2020 [19]. With the state-of-the-art network architecture, including CSP-Darknet53 backbone, SPP, PAN, and YOLOv3, YOLOv4 optimizes the accuracy and

high performance. YOLOv4-416 got 38 FPS and 62.8% mAP, better than YOLOv3-416 with 35 FPS and 55.3% mAP. YOLOv4-Tiny [20], using CSPDarknet53-tiny backbone, and FPN instead of SPP and PAN, provided extreme performance at 270 FPS with 38.1% mAP. The other proposed by Zicong Jiang [21] achieved at 294 FPS with 38.0% mAP. Glenn Jocher et al. introduced YOLOv5 in June 2020 [22], achieving 140 FPS. However, an official paper on the offered model has not yet been available. YOLOv5 uses CSP and PA-NET networks, along with enhancements including mosaic data augmentation, merging multiple images into one set of ratios for training, and automatic learning anchor boxes. Delong Qi et al. introduced YOLO5Face in May 2021 [23]. The author modified the architecture to support face detection with large and small size and landmark supervision.

3 Proposed System

This section introduces the proposed system. Section 3.1 presents the system design, and Sect. 3.2 describes the implementation of the proposed model.

3.1 System Design

The proposed system architecture includes three layers: Cloud, Edge, and End Devices, as shown below (Fig. 1):

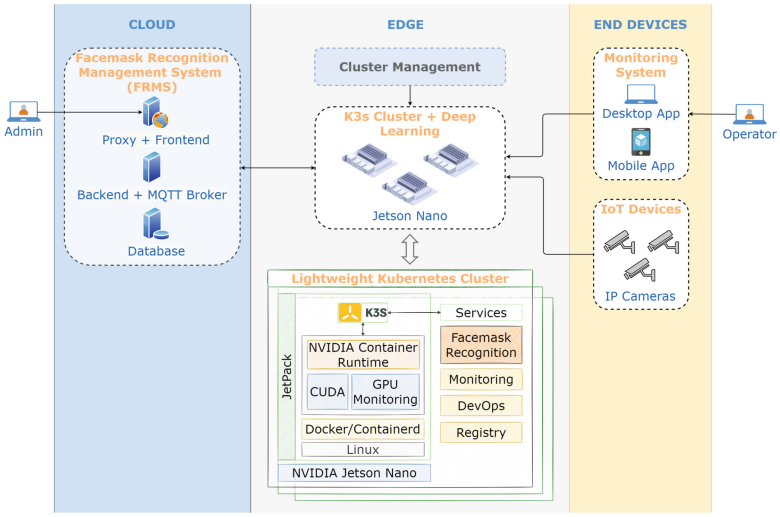

Fig. 1. System architecture.

3.1.1 Cloud

Facemask Recognition Management System based on MaskCam [24] covers these components: backend, frontend, proxy, MQTT broker and database (Fig. 2).

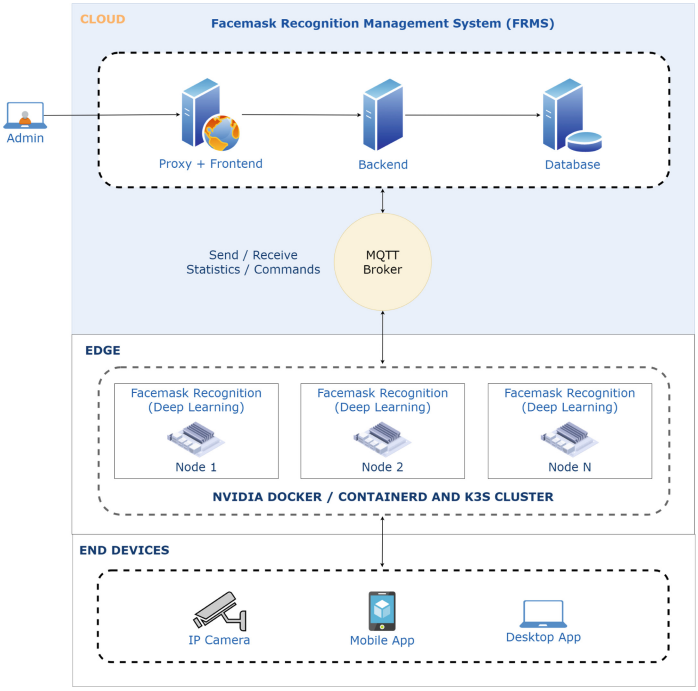

Fig. 2. FRMS architecture.

The backend comprises APIs using FastAPI, SQLAlchemy, Paho MQTT, etc. It receives information from facemask recognition services at the edge through MQTT broker running Eclipse Mosquitto, then saves to database and provides information to the frontend using Streamlit, Plotly, Paho MQTT, etc. The dashboard allows administrators to observe face mask recognition information as visual charts or manage facemask recognition services by sending requests through MQTT broker, including update status; start or stop streaming; capture a video. It also provides a function to download saved videos. Nginx acts as a HTTPs reverse proxy for the frontend. MQTT broker transports messages between FRMS and facemask recognition services. Database running PostreSQL stores statistical data and device information.

3.1.2 Intelligent Edge

The edge system runs K3s cluster, combining multiple master and worker nodes, to provide high availability, load balancing, scaling, and automated deployment. NVIDIA Docker/Containerd supports GPU to speed up facemask recognition services running modern lightweight YOLO models, etc. The cluster gives external access to its services via Nginx/Traefik Ingress Controller, Load Balancer, NodePort. Services within the cluster communicate with each other through ClusterIP. Theses services include cluster management, monitoring, registry, DevOps, and facemask recognition.

- **Cluster Management:** Rancher platform provides a centralized interface via UI, CLI and API to manage the edge system, and integrates Longhorn storage with no single point of failure (data replication across multiple nodes and recurring snapshot and backups), monitoring service, etc. (Fig. 3).

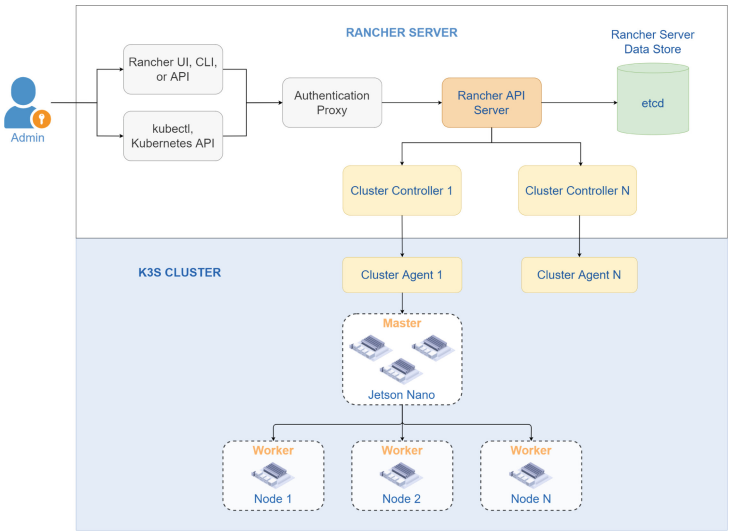

Fig. 3. Cluster management architecture.

- **Monitoring:** Prometheus, Grafana and AlertManager integrated with the cluster management system to observe the edge system (Fig. 4).

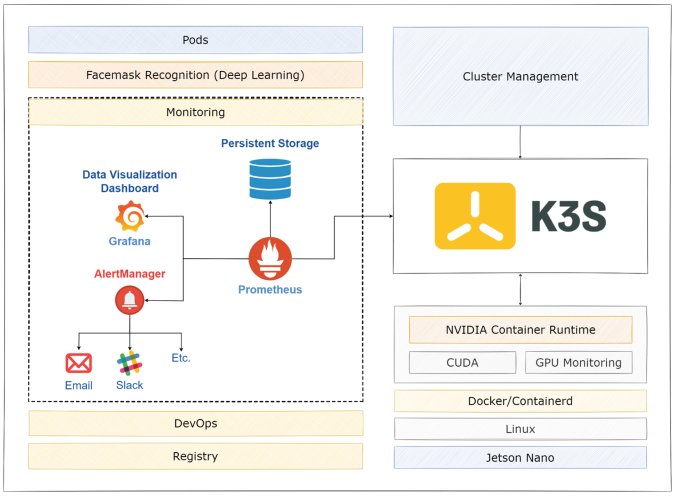

Fig. 4. Monitoring architecture.

- **DevOps:** GitLab DevOps platform, which incorporates Git repository, CI/CD management, etc. GitLab Runner works with GitLab CI/CD to run jobs. GitLab supports different Executors, including Docker, Kubernetes, SSH, Shell, etc. Meanwhile, Kubernetes Executor is our choice because it executes jobs on Pods of the cluster. The Pods end when finishing the jobs to free up resources. In this design, we apply CI/CD for facemask recognition services (Fig. 5).

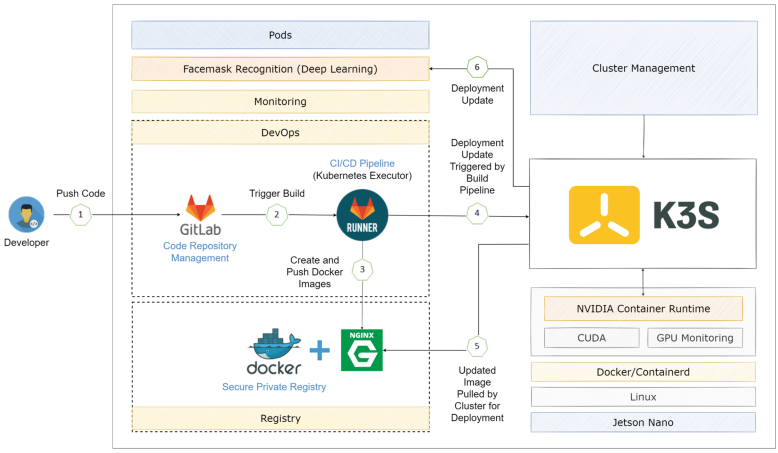

Fig. 5. CI/CD architecture.

- **Registry:** Docker Registry manages container images and Docker Credential Pass stores access information. Nginx with SSL/TLS as a reverse proxy. It sits in front of Docker Registry and forwards client requests to that application (Fig. 6).

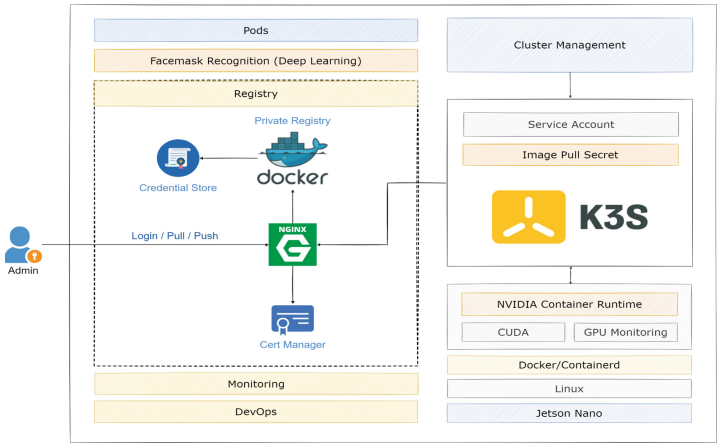

Fig. 6. Registry architecture.

- **Facemask Recognition:** Its design includes the orchestrator, facemask recognition inference, streaming server, and file server. The orchestrator coordinates among the processes. The inference uses lightweight YOLO object detection models (Fig. 7).

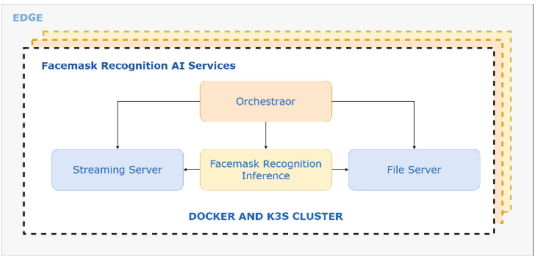

Fig. 7. Facemask recognition architecture.

3.1.3 End Devices

The IoT devices include ESP32, OV2640 and IMX219-160 cameras, etc. The application is written in C/C++. The mobile application developed with Flutter, Dart and NoSQL Google Firebase (Fig. 8).

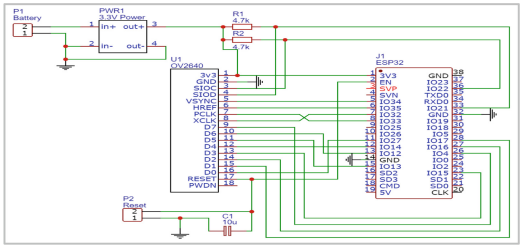

Fig. 8. Circuit design.

For COVID-19 design context, facemask recognition applications deploy in public areas, such as airport, schools, and so on (Fig. 9).

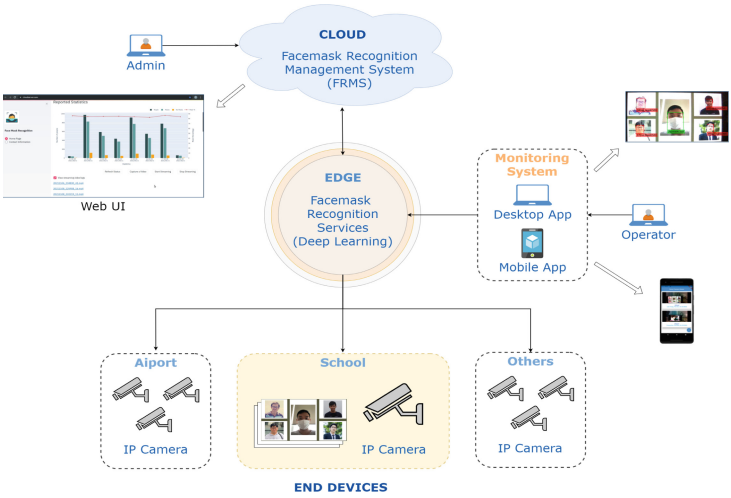

Fig. 9. Design context.

3.2 System Implementation

In this section, we present the implementation of the proposed system, including FRMS application; cluster management integrating storage and monitoring services; CI/CD pipelines for facemask recognition; IoT, mobile and desktop application. To make it at ease, we also developed a utility to automate the deployment of the proposed system. This will be more beneficial for deployment on a large scale; saving time and avoiding manual installation mistakes.

3.2.1 Cloud

See Fig. 10.

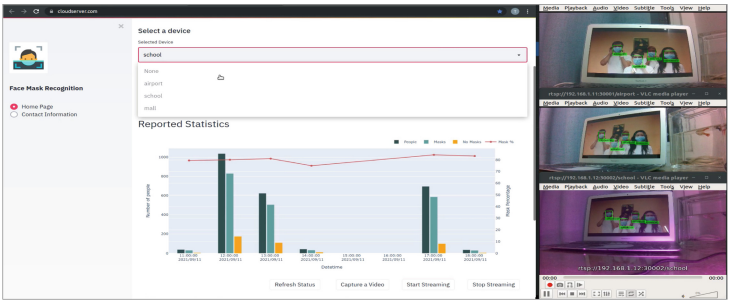

Fig. 10. FRMS application.

3.2.2 Intelligent Edge

See Figs. 11, 12, 13, 14 and 15.

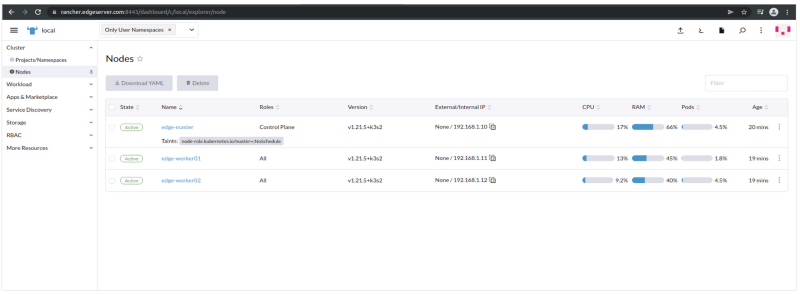

Fig. 11. Cluster management.

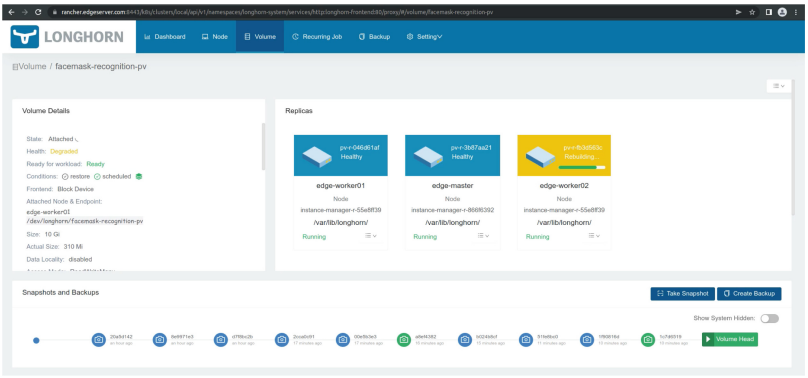

Fig. 12. Volume replicas, snapshots and backups.

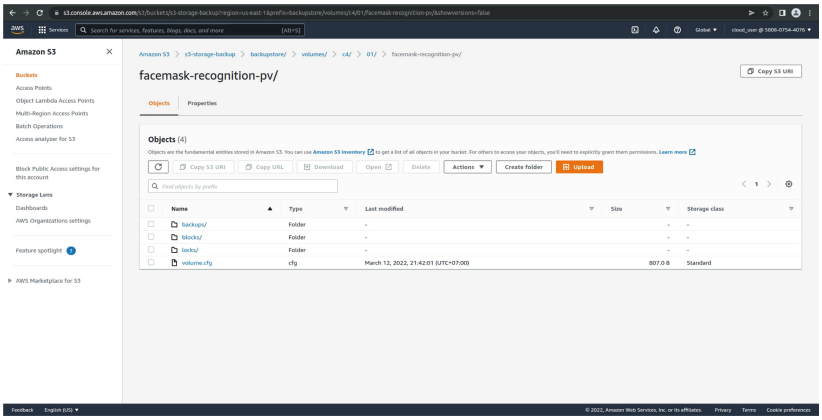

Fig. 13. Secondary data backup on Amazon S3.

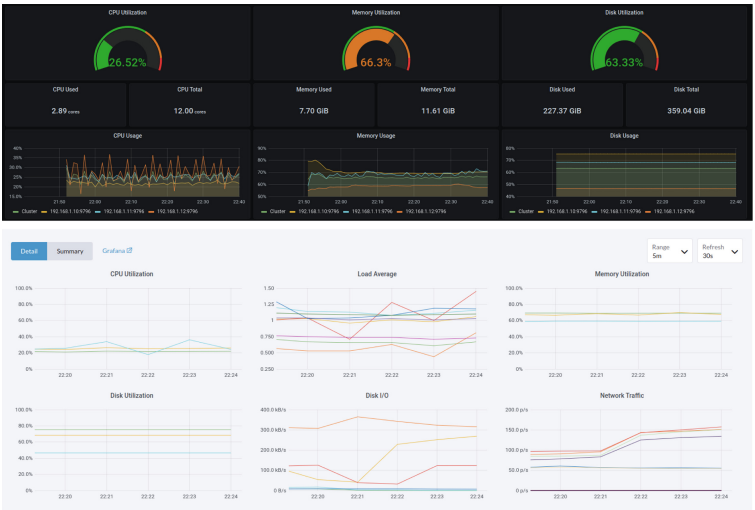

Fig. 14. Monitoring.

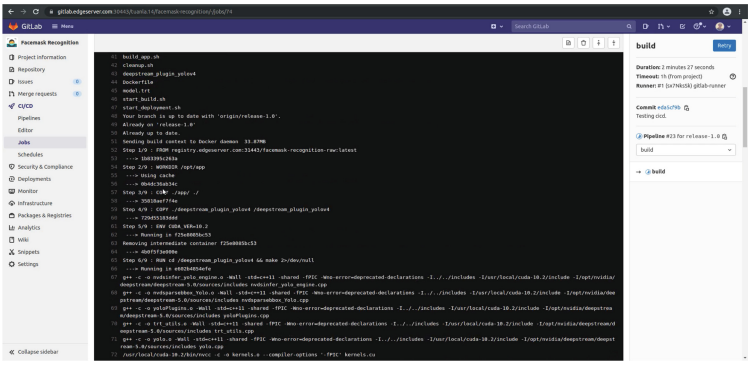

Fig. 15. CI/CD pipelines.

3.2.3 End Devices

See Figs. 16, 17 and 18.

Fig. 16. IoT application.

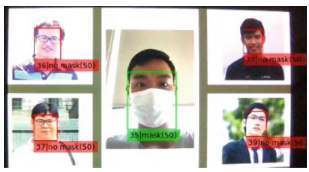

Fig. 17. Desktop app (VLC).

Fig. 18. Mobile app.

4 Experimental Results

This section covers three parts: Sect. 4.1 describes the accuracy assessment and performance analysis of facemask recognition deep learning models. Section 4.2 presents the availability evaluation of the edge services. Section 4.3 shows the hardware resource assessment occupied by the edge system.

4.1 Deep Learning Model Evaluation

In this experiment, we evaluated facemask recognition YOLO models on Jetson Nano (see Table 1). The datasets for the test include: The initial dataset of 920 images (training: 700, validation: 100, test: 120) provided by Kaggle [25] classified "mask" and "no_mask". The second dataset of 9,213 images (training: 5,533, validation: 1840, test: 1840) classified "mask", "no_mask" and "incorrect_mask". This dataset combined the first and second dataset [26] and new data. To get it more reliable, we used LabelImg [27] to fix incorrect labels for all the images; developed Python scripts to remove duplicate, missing data and formalize with the same YOLO annotation format; balance data among training, validation, and test (Fig. 19).

Table 1. Hardware specification.

No.	Role	Description
1	Master	Jetson Nano: GPUNVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores, CPU Quad-core ARM Cortex-A57 MPCore processor, RAM 4 GB 64-bit LPDDR4, 1600 MHz 25.6 GB/s, SD Card 64 GB
2	Worker	

Table 2. Deep learning model evaluation.

Model	mAP – validation			mAP – testing		
	IoU 0.25	IoU 0.5	IoU 0.75	IoU 0.25	IoU 0.5	IoU 0.75
YOLOv3 416 × 416	88.6	77.9	29.5	81.7	75.9	32.8
YOLOv3-Tiny 320 × 320	73.3	64.1	15.5	76.8	56.1	15.3
YOLOv3-Tiny 416 × 416	83.8	73.3	23.5	83.1	65.5	22.0
YOLOv3-Tiny 640 × 640	88.1	83.0	34.1	88.1	78.5	32.9
YOLOv3-Tiny 1024 × 576	88.7	81.9	26.7	90.1	85.2	38.5
YOLOv4 416 × 416	92.6	87.9	42.8	93.7	91.1	50.3
YOLOv4-Tiny-3L 320 × 320	88.0	80.6	33.3	86.8	78.4	35.9

(*continued*)

Table 2. (*continued*)

Model	mAP – validation			mAP – testing		
	IoU 0.25	IoU 0.5	IoU 0.75	IoU 0.25	IoU 0.5	IoU 0.75
YOLOv4-Tiny-3L 416 × 416	88.0	83.3	34.6	89.1	85.7	41.0
YOLOv4-Tiny-3L 640 × 640	89.0	85.7	35.6	89.0	87.0	38.5
YOLOv4-Tiny-3L 1024 × 576	88.6	85.0	32.9	90.8	89.4	40.7
YOLOv4-Tiny 320 × 320	72.6	67.7	28.0	64.2	57.0	24.0
YOLOv4-Tiny 416 × 416	85.3	79.8	36.4	75.4	72.3	37.6
YOLOv4-Tiny 640 × 640	89.4	86.0	41.5	89.2	85.9	48.9
YOLOv4-Tiny 1024 × 576	90.1	86.7	39.9	90.9	89.4	46.2

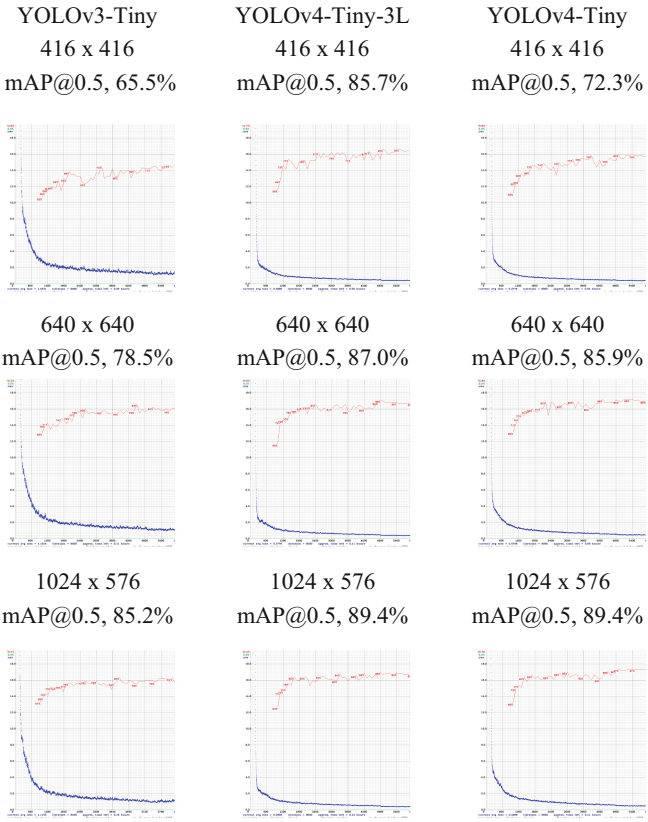

Fig. 19. Deep learning model evaluation.

According to Table 2, it shows that YOLOv4-Tiny-640 and YOLOv4-Tiny-1024 got 85.9% and 89.4% mAP with IoU 0.5. Besides, YOLOv4-Tiny-3L-1024 achieved 89.4% mAP with IoU 0.5 but 40.7% mAP with IoU 0.75, less than YOLOv4-Tiny-640 and YOLOv4-Tiny-1024 models. These results arrived from the initial dataset [25]. Next, we assessed the YOLO models with high accuracy, including YOLOv4-Tiny-3L-1024, YOLOv4-Tiny-640 and YOLOv4-Tiny-1024 on the second dataset. In Table 3, it shows YOLOv4-Tiny-1024 got the highest score with mAP@0.5 83.5% on test data and 85.8% on validation (Fig. 20).

Table 3. Deep learning model evaluation.

Model	mAP – validation			mAP – test		
	IoU 0.25	IoU 0.5	IoU 0.75	IoU 0.25	IoU 0.5	IoU 0.75
YOLOv4-Tiny-3L-1024	85.5	83.0	49.5	84.7	82.6	49.2
YOLOv4-Tiny-640	85.5	83.3	54.0	85.4	83.0	52.8
YOLOv4-Tiny-1024	87.9	85.8	52.3	86.0	83.5	52.4

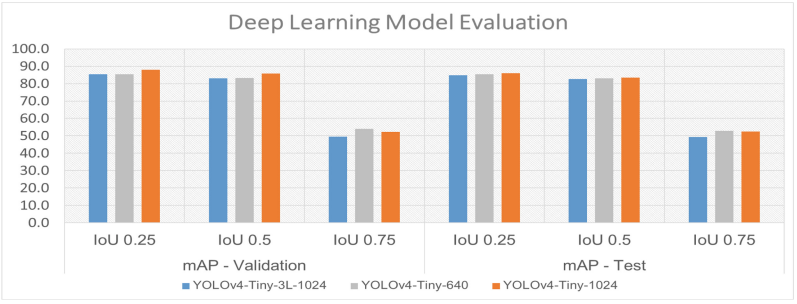

Fig. 20. Deep learning model evaluation.

Following Table 4, it shows YOLOv4-Tiny-640 and YOLOv4-Tiny-1024 models did 16.1 and 23.1 BFLOPS, less computation than YOLOv4-Tiny-3L-1024 with 27.3 BFLOPS. Therefore, they would be the choices for face mask recognition application.

Table 4. Comparison of BFLOPS for various YOLOv4-Tiny models.

Model	BFLOPS
YOLOv4-Tiny-3L-1024	27.3
YOLOv4-Tiny-640	16.1
YOLOv4-Tiny-1024	23.1

For the performance assessment, we conducted the test with YOLOv4-Tiny-1024 and a camera speed of 30 FPS. As a result, facemask recognition service using YOLOv4-Tiny-1024 achieved 13.7, 27.86 and 29.94 FPS with the inference interval of 0, 1 and 2 respectively in 10-W mode with a peak GPU speed of 921.6 MHz. The results show the inference interval of 2 gained the best performance. However, with a maximum GPU speed of 614.4 MHz in 5-W mode, it still achieved a relative performance of 27.16 FPS but less energy consumption (Table 5 and Fig. 21).

Table 5. Facemask recognition performance evaluation.

	Inference interval		
	0	1	2
Performance (FPS)	13.7	27.86	29.94

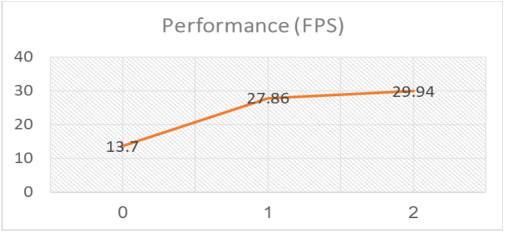

Fig. 21. Face mask recognition performance evaluation.

4.2 High Availability and Performance Evaluation

In this section, we assessed the high availability of edge services on K3s cluster, including registry, DevOps, and facemask recognition. Table 1 presents the hardware specification under the test. From the kubectl tool, we tried to delete a Pod of these services. As a result, the Pod terminated, and the others were in charge to avoid the downtime. In parallel, K3s cluster created a new Pod as a replacement for the deleted one. For another test, we evaluated the edge platform performance. Table 6 shows how long K3s cluster took to create or remove the edge services.

4.3 Hardware Resource Usage Assessment

In this part, we evaluated the hardware resource usage, especially the GPU speeding up face mask recognition service at the edge (Table 7 and Fig. 22).

The results show that the average GPU resource increased from 0.00% to 93.92%, 94.82%, 71.73% after the face mask recognition AI application in operation corresponding to the interval inferences of 0, 1 and 2. The GPU average difference was 86.82%. The average percentage of CPU and memory usage increased by 15.26% and 1018.70 MB.

Table 6. Edge platform performance assessment.

Service	Time	Creation/Removal
DevOps (GitLab, GitLab Runner)	<30 s	Persistent Volume, Persistent Volume Claim, Deployment, Service, Secret, etc.
Registry (Docker Registry, Nginx)		
Facemask Recognition (YOLO)		

Table 7. Hardware resource usage.

Component	Before	After			Difference
		0	1	2	
CPU (%)	19.12	35.12	35.73	32.27	15.26%
GPU (%)	0.00	93.92	94.82	71.73	86.82%
Memory (MB)	1,538.20	2,550.36	2,546.48	2,573.86	1,018.70

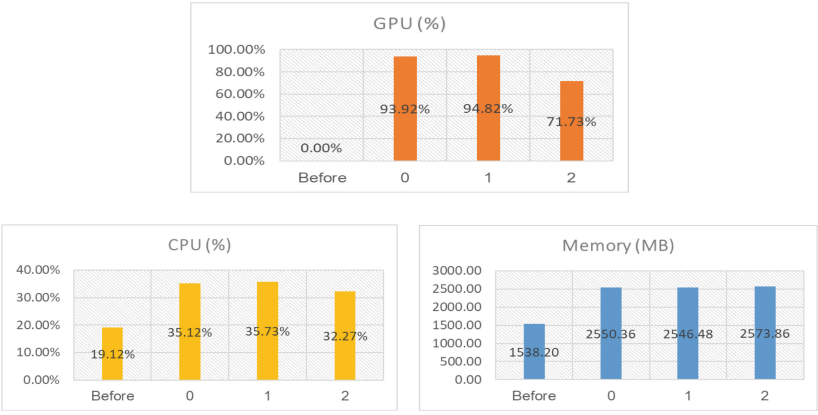

Fig. 22. Hardware resource usage chart.

5 Conclusion

In this paper, we introduced an intelligent edge system to assist with managing and developing microservices based on AI applications on embedded computers with hardware constraints. In this study, we implemented the face mask recognition application on the proposed system. For future work, we plan to investigate new features to expand the ecosystem of the proposed system; develop new functions for the face mask recognition application; explore new deep learning models capable of accuracy enhancements with optimized speed on embedded systems with limited hardware resources; investigate and develop a software platform to provide fast development to edge AI applications in computer vision.

Acknowledgement. This research is funded by the Ho Chi Minh City University of Information Technology, Vietnam National University, under grant number D1-2022-02.

References

1. Cisco annual internet report (2018–2023) white paper. Cisco. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html
2. Lasse Lueth, K.: State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time. IoT Analytics, 19 November 2020. https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/
3. Koon, J.: How AI Changes the Future of Edge Computing. EE Times Europe, 24 June 2019. https://www.eetimes.eu/how-ai-changes-the-future-of-edge-computing/
4. Morabito, R.: Virtualization on Internet of Things edge devices with container technologies: a performance evaluation. IEEE Access **5**, 8835–8850 (2017). https://doi.org/10.1109/ACCESS.2017.2704444
5. Le-Anh, T., Ngo-Van, Q., Vo-Huy, P., Huynh-Van, D., Le-Trung, Q.: A container-based edge computing system for smart healthcare applications. In: Vo, N.-S., Hoang, V.-P., Vien, Q.-T. (eds.) INISCOM 2021. LNICSSITE, vol. 379, pp. 324–336. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77424-0_27
6. Sanjaya, S.A., Adi Rakhmawan, S.: Face mask detection using MobileNetV2 in the era of COVID-19 pandemic. In: 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI), pp. 1–5, October 2020. https://doi.org/10.1109/ICDABI51230.2020.9325631
7. Suresh, K., Palangappa, M., Bhuvan, S.: Face mask detection by using optimistic convolutional neural network. In: 2021 6th International Conference on Inventive Computation Technologies (ICICT), pp. 1084–1089, January 2021. https://doi.org/10.1109/ICICT50816.2021.9358653
8. Jiang, X., Gao, T., Zhu, Z., Zhao, Y.: Real-time face mask detection method based on YOLOv3. Electronics **10**(7) (2021). Article no. 7. https://doi.org/10.3390/electronics10070837
9. Böhm, S., Wirtz, G.: Profiling lightweight container platforms: MicroK8s and K3s in comparison to Kubernetes, March 2021
10. Elmoulat, M., Debauche, O., Saïd, M., Mahmoudi, S., Manneback, P., Lebeau, F.: Edge computing and artificial intelligence for landslides monitoring. Procedia Comput. Sci. **177**, 480–487 (2020). https://doi.org/10.1016/j.procs.2020.10.066
11. Debauche, O., Saïd, M., Elmoulat, M., Mahmoudi, S., Manneback, P., Lebeau, F.: Edge AI-IoT pivot irrigation, plant diseases and pests identification. Procedia Comput. Sci. **177**, 40–48 (2020). https://doi.org/10.1016/j.procs.2020.10.009
12. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. arXiv:1506.01497 Cs, January 2016. http://arxiv.org/abs/1506.01497
13. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. arXiv:1703.06870 Cs, January 2018. http://arxiv.org/abs/1703.06870
14. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
15. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. arXiv:1506.02640 Cs, May 2016. http://arxiv.org/abs/1506.02640

16. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. arXiv:1612.08242 Cs, December 2016. http://arxiv.org/abs/1612.08242
17. Redmon, J., Farhadi, A.: YOLOv3: an incremental improvement. arXiv:1804.02767 Cs, April 2018. http://arxiv.org/abs/1804.02767
18. Mao, Q.-C., Sun, H.-M., Liu, Y.-B., Jia, R.-S.: Mini-YOLOv3: real-time object detector for embedded applications. IEEE Access **7**, 133529–133538 (2019). https://doi.org/10.1109/ACCESS.2019.2941547
19. Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y.M.: YOLOv4: optimal speed and accuracy of object detection. arXiv:2004.10934 Cs Eess, April 2020. http://arxiv.org/abs/2004.10934
20. Bochkovskiy, A.: Darknet (2021). https://github.com/AlexeyAB/darknet
21. Jiang, Z., Zhao, L., Li, S., Jia, Y.: Real-time object detection method based on improved YOLOv4-tiny. arXiv:2011.04244 Cs, December 2020. http://arxiv.org/abs/2011.04244. Accessed 04 Aug 2021
22. Nelson, J.: YOLOv5: state-of-the-art object detection. Roboflow Blog, 10 June 2020. https://blog.roboflow.com/yolov5-is-here/
23. Qi, D., Tan, W., Yao, Q., Liu, J.: YOLO5Face: why reinventing a face detector. arXiv:2105.12931 Cs, May 2021. http://arxiv.org/abs/2105.12931
24. MaskCam. BDTI (2021). https://github.com/bdtinc/maskcam
25. Purohit, A.: Face mask dataset with YOLO format. https://kaggle.com/aditya276/face-mask-dataset-yolo-format
26. ethancvaa: Properly-Wearing-Masked-Detect-Dataset (2021). https://github.com/ethancvaa/Properly-Wearing-Masked-Detect-Dataset
27. Tzutalin, LabelImg (2021). https://github.com/tzutalin/labelImg