JURNAL NASIONAL TEKNIK ELEKTRO DAN TEKNOLOGI INFORMASI
Volume 13 Number 1 February 2024

Application of Computation Offloading in Edge Computing-Based Level Crossing Violation Detection Systems

Rian Putra Pratama[1], **Suhono Harso Supangkat**[2]

[1] Center for Smart Mechatronics, National Research and Innovation Agency, Bandung, Indonesia
[2] School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Bandung, Indonesia

[Received: 28 July 2023, Revised: 13 November 2023, Accepted: 18 December 2023]
Corresponding Author: Rian Putra Pratama (email: rian011@brin.go.id)

ABSTRACT — Level crossings remain a problem in several cities due to high violations. Currently, surveillance at level crossings is still performed conventionally. Since problems at level crossings are increasingly complex and conventional solutions are no longer effective, an intelligent video surveillance system is necessary. Intelligent video surveillance system implementation is a complex task and requires devices with extensive computing resources. This research aims to optimize the system for processing data in real-time by conducting computation near the data source and dividing computing tasks across several edge devices. This research proposes a solution in the form of an edge computing-based intelligent video surveillance system with a computation offloading method on limited devices. This research has two development stages. The initial stage involved developing an object detection model using a dataset of level crossings in Bandung City. The second stage was developing an edge computing-based system by applying the computation offloading method on limited computing devices. The edge computing method extends cloud computing to the network's edge, enabling calculations near the data source. Conversely, the computation offloading method improves edge computing performance by dividing computing tasks. Results showed an increase in computing speed of around 1.5 times faster, with a violation detection accuracy rate reaching 89.4%. Additionally, GPU temperature decreased by 5.50 °C, GPU usage decreased by 44.05%, memory usage decreased by 301 Mb, and power consumption decreased by 2.28 W. The system developed is effective and efficient in optimizing the performance of the violation detection system in level crossings on limited computing devices.

KEYWORDS — Computation Offloading, Edge Computing, Violation Detection Systems, Level Crossings, Intelligent Video Surveillance Systems.

I. INTRODUCTION

A level crossing is a junction between a railway line and a highway [1]. The Directorate General of Land Transportation of the Republic of Indonesia has emphasized the importance of monitoring security and safety at level crossings. Problems at level crossings are generally caused by a relatively high level of violations and lack of surveillance, thereby elevating the accident risk. Violations generally occur when motorists break a crossing gate that has been closed [2]. Problems at level crossings are increasingly complex, and conventional solutions are no longer effective in resolving these problems. Therefore, a technological approach to monitoring systems at level crossings is crucial to reduce traffic violations and potential accidents [3]. Intelligent monitoring systems at level crossings are part of innovative city development from a safe and secure aspect [4].

Currently, intelligent video surveillance systems utilizing computer vision are able to integrate multiple image and video analysis algorithms [5]. The application of intelligent video surveillance systems in innovative city development through a computer vision approach has grown significantly over the last decade [6]. The application of intelligent video surveillance system technology encompasses traffic control, road activity detection, and behavior monitoring of road users [7]. Research on intelligent video surveillance systems seeks to replace human operators or conventional systems with video processing algorithms capable of performing tasks autonomously [8].

The current state of the art of intelligent video surveillance systems is focused on three main tasks: detection, tracking, and activity recognition or understanding of certain behaviors and situations using neural networks in deep neural networks (DNN) algorithms [9]. Detecting, tracking, and recognizing activities using DNN is complex and requires devices with considerable computing resources [10]. Implementing DNN in cloud computing requires significant time and computational load. Cloud computing receives all video data uploaded from the Internet of things (IoT) devices in the form of camera sensors and performs computations, resulting in increased costs for data transmission on the network, including latency, bandwidth, and computing resources [11].

From this, the edge computing paradigm emerged to overcome the problem of rising data transmission costs in networks in cloud computing [12]. Edge computing performs the computing process closer to the data source so that communication latency, bandwidth, and resources can be reduced [13], [14]. Edge computing is conducted to expand cloud computing to the network's edge to perform computational calculations in close proximity close to the data source, namely on the edge device [15]. Through the utilization of edge computing, data processing can be done locally at the network's edge, resulting in a reduction of computing load and the efficient execution of activities without dependence on cloud infrastructure.

The primary challenge in implementing intelligent video surveillance systems on edge devices is the limitation of computing resources. Limited computing devices have disadvantages in the form of limitations in terms of processing capacity or computing power. These limitations encompass several aspects, including processing speed, memory capacity, and the ability to handle tasks requiring high computing power. Optimization is required to attain ideal conditions for creating effective and efficient solutions, one of which is the computation offloading method [16]. Computation offloading is a method to improve edge computing performance by dividing computing tasks into devices with more significant computing resources to overcome limited computing resources on edge devices [17].

Research on intelligent video surveillance systems at level crossings has been conducted in various countries. A study has developed a computer vision-based level crossing monitoring system in the State of New Jersey and the city of Ashland, Virginia, United States. This system utilized the region of interest (ROI) method and regions with convolutional neural networks (R-CNN) mask object identification model [18]. Reference [19] has developed a technological solution for monitoring and making intelligent decisions at level crossings using MobileNet architecture and CNN algorithms for object classification and obstacle detection. Other research in the Czech has conducted detection and classification at level crossings and train warnings using the YOLOv3 deep neural network (DNN) model implemented on edge devices [20].

This research developed an edge computing-based level crossing monitoring system that could detect violations by developing object detection and recognition models and optimizing computing performance using the computation offloading method on limited computing devices. In the initial stage, an object detection and recognition model were established using an object detection algorithm, referring to previous research [21], [22]. In the second stage, the results of model formation were applied to devices with limited computing resources using the computation offloading method to accelerate inference time and reduce the computational load [23]. In the third stage, evaluations and test scenarios were conducted to measure the effectiveness and efficiency of the system prototype [24].

II. METHODOLOGY

This research refers to previous research that designed monitoring system technology for level crossings. Although previous research applied the edge computing approach to the developed system architecture, its implementation on a single-edge device had limited resources, including limited memory and computational processing capabilities.

The architectural design which was developed in this research consisted of three layers, namely the end device layer, the edge layer, and the cloud service layer. The development process involved two stages. In the initial stage, an object detection model was developed to recognize objects in the level crossing environment by collecting a dataset at one of the level crossings in Bandung City. In the second stage, an edge computing-based system was developed by implementing computation offloading on devices with limited computing resources, namely dividing the computing process into two edge devices.

Testing in this research was performed using two scenarios in order to compare computing performance on limited computing devices. The first scenario used a system design from previous research, while the second scenario applied the computation offloading method developed in this research.

A. MODEL FORMATION

The system development process in this research was initiated by a model formation process flow, which involved the following steps: dataset collection, model training, and model evaluation [25]. The aim of establishing this model is to detect and recognize violation objects in the level crossing area, which is implemented on limited computing devices. The model formation process began with the data preparation stage. Raw data were converted into data in a more structured format and ready for use. Next, the data-sharing process was divided into three parts. The first part, namely the training data, is a subset of the data used to train the predictive model. Second, validation data are a subset of data used to evaluate model performance during the training process. Meanwhile, test data are a separate subset of data that is not used during the training and validation process. These data are used to test the performance of the trained model against new data that have not been seen before. The stages of model formation in this research are explained as follows.

1) DATASET COLLECTION

The dataset collection process began with recording a video at one of the level crossings in Bandung City. From this process, 3,000 images were acquired. These images consisted of 21,000 annotations from six classes, with detailed annotations of 8,638 motorbikes, 6,150 cars, 2,089 trucks, 1,288 doorstops, 1,025 public transportation, and 83 trains. As shown in Figure 1, the image dataset was divided into three parts with a ratio of 7:2:1. Specifically, 70% of the dataset was allocated for training data, which amounted to 2,200 data; 20% for validation data, which amounted to 655 data; and 10% for testing data, which amounted to 324 data. The following process is to resize the image to sizes of 416 pixels and 640 pixels, taking into account model training time and based on several other pixel size configurations.

2) MODEL TRAINING

At the model training stage, the collected dataset was trained using the CSPDarknet53 CNN architecture on YOLOv5s using the PyTorch library contained in the custom_yolov5s.yaml file [26]. YOLOv5s is one of the most popular algorithms in object detection. According to researchers at AI Research, the unified architecture of YOLOv5s is very simple. A single convolution network allows YOLOv5s to directly detect objects by only passing through the neural network once [27]. YOLOv5s consists of three parts, namely backbone, neck, and head. The backbone acts as a feature extractor, the neck acts as a feature aggregator, and the head is responsible to perform localization and classification on each bounding box. The backbone extracts features using BottleNeckCSP and SPP, while the neck collects sample features. The results are in the head. YOLOv5s performs object classification with a convolutional network.

The training process used Python code with the train.py file in the YOLOv5 model library using data variations in image size, number of epochs, and batch size [28]. Training this model involved configurations with varying image sizes of 416 pixels and 640 pixels; epoch numbers of 50, 100, and 150; and batch sizes of 16, 32, and 64 data. Image size affects the model's ability to capture details, the epoch numbers determine the success rate of the model learning from the dataset, and batch

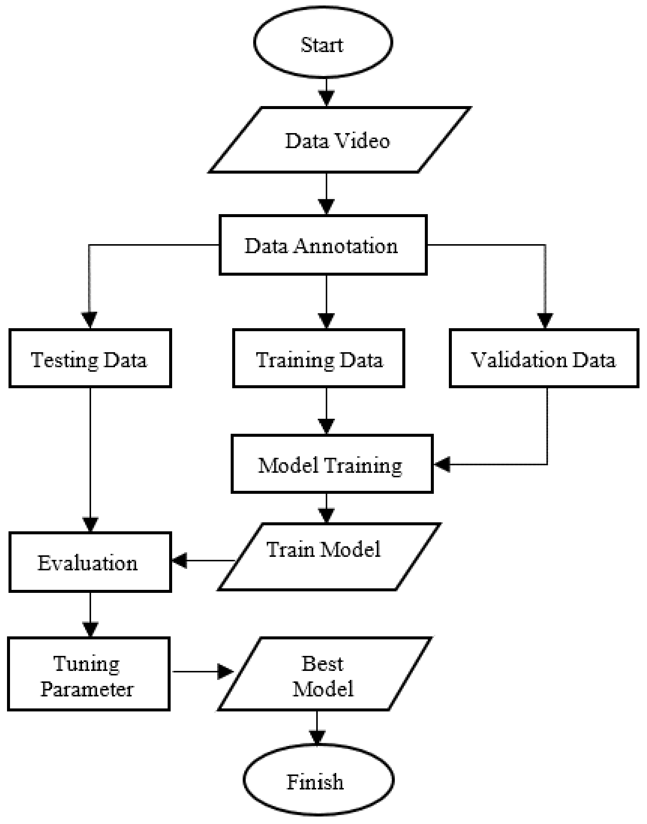


Figure 1. Model training flowchart.

size affects training efficiency. Experiments were conducted to find the most effective and efficient parameters to detect violation data objects at level crossings.

3) MODEL EVALUATION

At the evaluation stage, the model training process that had been conducted produced precision, recall, and mean average precision (mAP) values. The mAP metric was used to measure the average accuracy of each class. If the mAP value was not good enough, changing the parameter configuration in the model training process was necessary until a good mAP value was obtained. In this research, a model with the most optimal mAP value was obtained from the results of several variations in the configuration of the model training process. This model had a parameter configuration with image sizes of 640, batch sizes of 32, and epoch numbers of 100. From the model training results, as shown in Table I, two most optimal models were obtained. Model 1 achieved mAP@0.5 values of 0.993, while model 2 achieved mAP@0.5 values of 0.852. The mAP@0.5 metric is a metric that measures object detection performance at a confidence level of 0.5, indicating the success of the system in detecting objects with a prediction confidence value of 0.5 or higher [29].

Model 1 consisted of one class of level-crossing gate objects, functioning to recognize the movement of level-crossing gate objects. Model 2 comprised five classes: motorbikes, cars, trucks, public transportation, and train. Model 2 functioned to detect objects crossing level crossings. The results of this model training demonstrated that the model performed well in detecting gate and vehicle objects in level-crossing environments.

B. SYSTEM DEVELOPMENT

The flow of the system's development for identifying and categorizing violation objects at level crossings in this research is shown by the sequence diagram in Figure 2. The sequence diagram explains the process for detecting level crossing gate objects. If the level crossing gate was detected as being closed, node 1 sent a status message to node 2. Node 2 received the message and accessed the end device to retrieve video data. Next, at node 2, video data were used to perform object detection for each violation that occurred. The detection results were recorded and sent to the data logger or cloud service, provided that the gate object status message from node 1 continued to indicate that the gate was closed. However, if node 1 sent a message indicating that the gate object was open, the process on node 2 stopped. The computing process in this system used the message queuing telemetry transport (MQTT) communication protocol as a data communication medium between layers. MQTT utilizes a publish and subscribe communication model, which enables devices to subscribe to certain topics and receive messages published on those topics. This feature facilitates distributed communication and is effective for delivering media, including images or videos, to subscribed devices [30].

TABLE I
MODEL TRAINING RESULT

Parameter	Model 1	Model 2
Object	Railroad crossing gate	Motorbike, vehicle, truck, public transportation, and train
Parameter Configuration	img 640 batch 32 epoch of 100	img 640 batch 32 epoch of 100
Precision	0.988	0.808
Recall	0.965	0.751
mAP	0.993	0.852

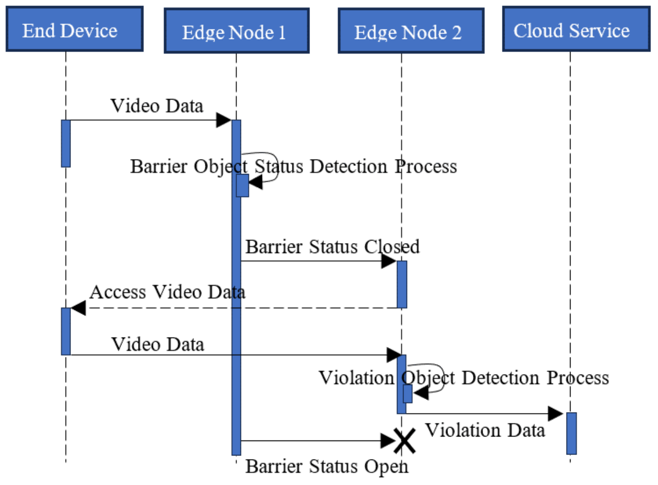


Figure 2. System development sequence diagram.

1) NODE 1 EDGE LAYER

As illustrated in Figure 3, The edge layer of node 1 used a Raspberry Pi 4 to receive real-time streaming protocol (RTSP) from video data at level crossings. The computational process at this layer involved a gate object detection model. The system algorithm detected the gate object, recognized the movement of the object, and tracked it with a bounding box. The centroid point's coordinates were updated by the system as the gate closed the crossing. The system marked a closed-level crossing situation if the centroid point crosses the specified linear line. Once the situation was detected, the edge device of node 1 communicated with node 2 to continue the computing process.

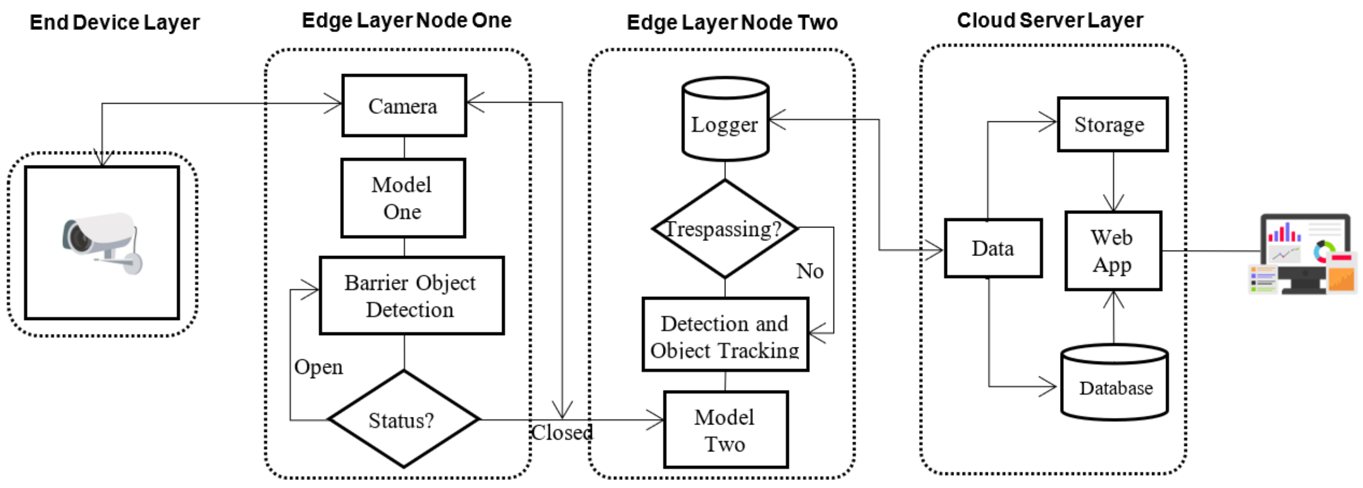


Figure 3. Flow of the computation offloading process.

In the communication process, the edge device of node 1 acted as an MQTT client (publisher) via the local network and offloaded it to the edge device of node 2. The process began with connecting to the broker via IP address and port. If connected, the system reads frames from video data. If the gate was detected as closed, each frame was converted to byte data type using the OpenCV imencode library, then published to the broker with the topic and encoded data. This process was repeated as long as the connection to the broker remained connected.

2) NODE 2 EDGE LAYER

At the edge node layer 2, NVIDIA Jetson Nano was used as an edge device [31]. The process is shown in Figure 3. The edge device of node 2 received a message from the edge node 1, indicating that the crossing gate closed since a train would pass. When the gate was open, the edge device status of node 2 became idle, so the computing process at this layer was only active when receiving a message from the edge device of node 1. The task involved multi-detection of vehicle objects and tracking vehicle movements. The violation area was determined by four coordinate points that form a polygon, and a violation was detected if the vehicle's centroid point was within this area. The system recorded each violation and sent the data in JSON format to the cloud layer via REST API.

The communication process on the edge device of node 2 initiated by making a connection to the broker. After successfully connecting, the edge device of node 2 received the data sent by the publisher, and then the received data were converted back to their original data form. Next, the decoding process was carried out to display the image. The data received in byte form were converted with the np.frombuffer function into ndarray and then converted back to the original data using the cv2.imdecode function to decode the data into images. An object detection process was conducted on the image data received to identify violations. Object detection results (violation data) were stored in JSON array format and ready to be sent to the cloud layer via the application programming interface (API). This communication flow allows edge devices of node 2 to communicate with the broker and send detected violation data to the cloud layer via API.

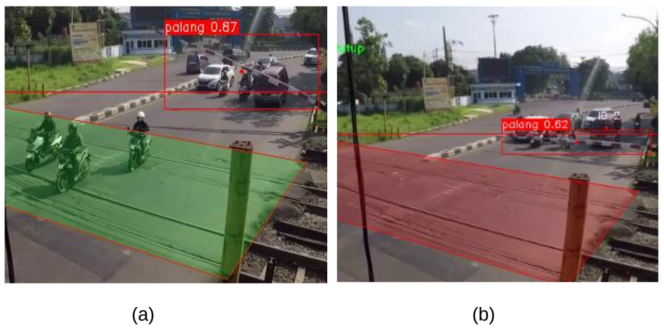
3) CLOUD SERVICE LAYER

The platform used at the cloud service layer was the Heroku cloud platform. The process stages at the cloud service layer are shown in Figure 3. The task of the cloud service layer was to receive any violation data in the form of an API, which was recorded by the system in the data logger. Within a certain period, all violation data in the logger were sent and stored on the cloud storage platform. Subsequently, violation data were analyzed for the purpose of monitoring the level of violations at level crossings. This information was used for decision-making related to the process of monitoring and controlling potential accidents and traffic violations at level crossings. Apart from that, the data were also used for planning the construction of nonlevel crossings [32].

C. SYSTEM IMPLEMENTATION

The implementation of this research used several devices that support the application of edge computing technology. Edge node 1 used a Raspberry Pi 4 device with a 64-bit Quad-Core Cortex-A72 1.5 GHz CPU specification and 4 GB memory. In contrast, edge node 2 used an NVIDIA Jetson Nano device with a Quad-Core ARM Cortex A57 processor specification and memory of 4 GB 128-core Maxwell. Meanwhile, the cloud server used was the Heroku cloud platform. The implementation of system development in this research was carried out using video data of the crossing conditions at JPL 156 km 152 Andir Station, Bandung City. The process of capturing the video data used a USB camera with a resolution of 1,028 × 780, which was placed in the side corner of the level crossing on a tripod at a height of around 3 m so that the camera could record the entire area of the level crossing.

Figure 4(a) shows the situation when the level crossing gate is still open. The system succeeded in detecting the crossing gate object, which was marked with a bounding box, and the position of the centroid point and the violation area were still green. This green color indicates that the passing vehicle was not detected as having committed a violation. Furthermore, in Figure 4(b), the level crossing gate bars are closed. The system detected the movement of the gate object via the centroid point. Assuming the centroid point was below the line, the gate status would be changed to "closed", and the violation area would turn to red, indicating that every passing vehicle object would be identified as having committed a violation. Finally, Figure 4(c) shows the situation when a violation occurs. The system succeeded in detecting and identifying violation objects in the violation area when the level crossing gate was closed.



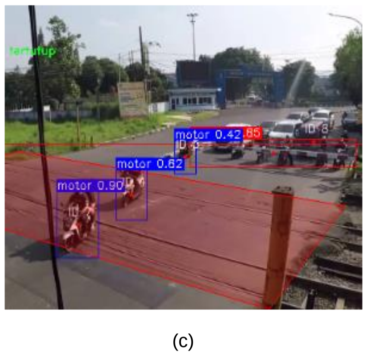


Figure 4. System implementation, (a) the gate is open, (b) the gate is closed and (c) the system detects a violation

III. RESULTS AND DISCUSSION

Testing in this research was conducted using two scenarios. The aim is to compare computing performance in each scenario so that the results of system development can be known, whether it can improve computing performance on limited computing devices in terms of accuracy, computing speed, and computing performance or not [33]. Scenario 1 referred to the system design that had been carried out in previous research [20], whereas scenario 2 implemented the computation offloading method to develop the system in the present study.

A. ACCURACY TESTING RESULTS

Accuracy testing was conducted using video data of level-crossing situations. The amount of video data used in testing was ten videos, with varying times and weather conditions. From the two accuracy testing scenarios, the results of the confusion matrix were used to measure the level of accuracy of the violation detection model at level crossings. True positive (TP) is a condition when the system detects an object that has committed a violation. The system should detect that the object has not committed a violation. False positive (FP) is a condition where the system incorrectly detects an object as a violation. False negative (FN) is where the object has committed a violation, but the system does not record it as a violation.

Figure 5 shows a graph comparing the accuracy values of the violation detection models. Scenario 2 obtained better results than Scenario 1. The recall value in scenario 2 was 3% better, the precision value in scenario 2 is 1% better, the *F*-score value in scenario 2 is 2.8% better, and the test accuracy level in scenario 2 was better by around 3.40%. The accuracy of the implemented violation object detection model was notably high at 89.4%. This finding suggests that the detection system has succeeded in recognizing and classifying the object of the violation well and successfully recorded it as a violation.

B. INFERENCE SPEED TESTING RESULTS

Inference speed refers to the speed of a system or model to produce predicted results or outputs after receiving input.

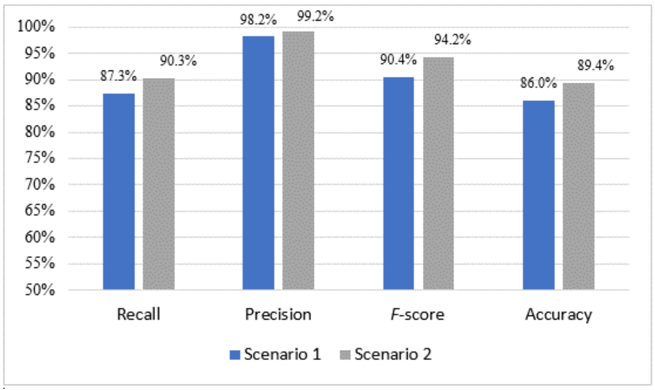


Figure 5. Comparison of accuracy models.

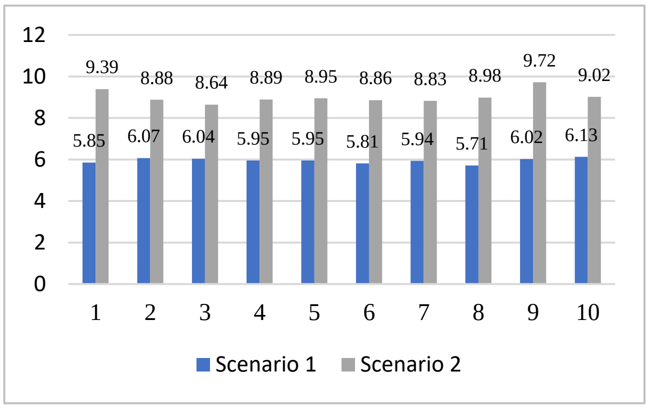


Figure 6. Comparison of computing speed.

Inference speed testing with ten tests using ten video data produced an average speed of 5.95 fps in scenario 1 and 8.95 fps in scenario 2. Based on this data, a comparison of test results is depicted in Figure 6. The graph comparing the average inference speed shows that Scenario 2 produces a higher average inference speed compared to scenario 1.

C. COMPUTING PERFORMANCE TESTING RESULTS

Computing performance refers to the ability of hardware to carry out given tasks with high efficiency and effectiveness [34]. Figure 7 shows the performance of the graphics processing unit (GPU) in the form of temperature. It is evident that the GPU temperature on the NVIDIA Jetson Nano device was impacted by the computation process when the violation detection process was run on the NVIDIA Jetson Nano device. As seen in the graph, GPU in scenario 2 had a lower temperature value compared to the GPU temperature in scenario 1. It happened because the computing process was performed only when the crossing gate was closed. The average GPU temperature was 36.85 °C in scenario 1 and 31.34 °C in scenario 2, or an average decrease of 5.50 °C. Figure 8 shows a graph of test results from using the GPU. The average value of GPU usage was obtained at 50.58% in scenario 1 and 6.53% in scenario 2, or a decrease of around 44.05%. The computing process in scenario 2 was more effective because the GPU was only used when the level crossing was closed. In Figure 9, memory usage data are displayed. The memory usage test results show that scenario 1 used 3,790 Mb of memory on average, while scenario 2 used 3,489 Mb. In other words, there was a decrease in memory usage of around 301 Mb. Meanwhile, a comparison graph of power consumption is shown in Figure 10. In scenario 1, the Jetson Nano device used an average power consumption of 5.83 W. Meanwhile, in scenario 2, the Jetson Nano used a power consumption of 3.55 W. There is a

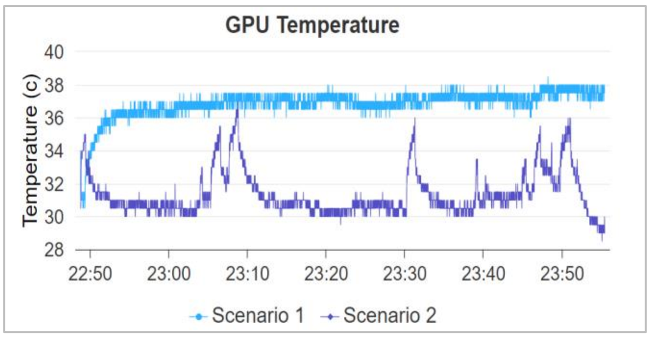


Figure 7. GPU temperature comparison results.

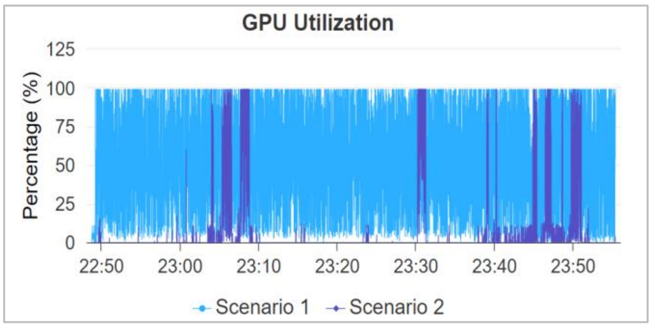


Figure 8. GPU utilization comparison results.

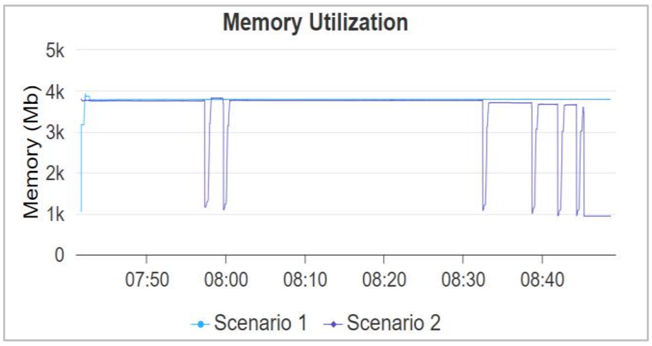


Figure 9. Memory utilization comparison results.

decrease in average consumption power of 2.28 W on the Jetson Nano device. In scenario 2, with the Raspberry Pi 4 device implemented as node 1, the average power consumption was 2.61 W, which was smaller than that on the Jetson Nano. It happened because the computing process on the Raspberry Pi 4 device only used the CPU and not the GPU.

The test results in Table II show that the application of computation offloading to the prototype edge computing-based violation detection system at level crossings yielded better results, as evidenced by the increase in computing performance. The application of computation offloading succeeded in increasing computing performance on the Jetson Nano limited computing devices. The average computing speed increases by about 1.5 times, demonstrating the efficiency of the computing load-sharing strategy. The accuracy of the implemented violation object detection model reached 89.4%, which is considered high. It indicates that the detection system has succeeded in recognizing and classifying violation objects. Apart from increasing speed, the application of computation offloading had succeeded in reducing the burden on the computing process on the Jetson Nano device. Reducing GPU temperature, GPU usage rate, memory usage rate, and power consumption provides an overview of resource usage efficiency. The system developed in this research proves the effectiveness and efficiency of implementing computation

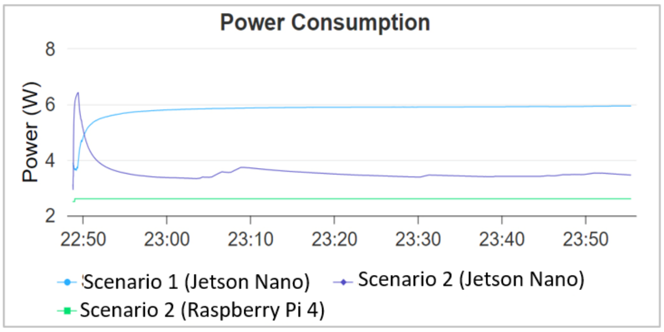


Figure 10. Power Consumption Comparison Results

TABLE II
COMPARISON OF TEST RESULTS

Testing	Scenario 1	Scenario 2	Delta
Accuracy	86.0 %	89.4 %	+3.4 %
Inference Speed	5.95 fps	8.95 fps	+3 fps
GPU Temperature	36.85 °C	31.34 °C	-5.50 °C
GPU Utilization	50.58 %	6.53 %	-44.05 %
Memory Utilization	3,790 Mb	3,489 Mb	-301 Mb
Power Consumption	5.83 W	3.55 W	-2.28 W

Information: Delta = scenario 2 – scenario 1

offloading in optimizing the performance of violation detection systems at level crossings on limited computing devices.

IV. CONCLUSION

The prototype violation detection system at level crossings based on edge computing, developed in this research, has succeeded in increasing computing performance by dividing the computing load by implementing computation offloading. From the developments carried out, the implementation of computation offloading succeeded in increasing the average computing speed by around 1.5 times faster on the Jetson Nano limited computing devices. Apart from that, the violation object detection model attained an accuracy rate of 89.4%. Apart from increasing speed, the application of computation offloading has also succeeded in reducing the burden on the computing process on the limited Jetson Nano computing device. It is evidenced from a decrease in GPU temperature of around 5.50 °C, a decrease in GPU usage of 44.05%, a decrease in memory use of 301 Mb, and a decrease in power consumption of 2.28 W. The data shows that the application of computation offloading to an edge computing-based level crossing violation detection system provides good results. Significant improvements in inference accuracy and speed, as well as reductions in GPU temperature, GPU usage, memory usage, and power consumption, demonstrate efficient use of resources on limited computing devices.

CONFLICTS OF INTEREST

The authors state that in the research entitled “Application of Computation Offloading in Edge Computing-Based Level Crossing Violation Detection Systems” has no conflict of interest.

AUTHORS’ CONTRIBUTIONS

Conceptualization and methodology, Rian Putra Pratama; software, Rian Putra Pratama; data acquisition, Rian Putra Pratama; data analysis, Rian Putra Pratama; writing—

preparation of the original draft, Rian Putra Pratama; supervision and review, Suhono Harso Supangkat.

ACKNOWLEDGMENT

Thanks are expressed to the parties who have supported the author in carrying out this research, including the Center for Smart Mechatronics of the National Research and Innovation Agency, School of Electrical Engineering and Informatics of Institut Teknologi Bandung, and the domestic scholarship program of the Ministry of Communication and Information of the Republic of Indonesia.

REFERENCES

[1] "Lalu Lintas dan Angkutan Jalan," Undang-Undang Republik Indonesia No. 22, 2009.

[2] "Buku Statistik Bidang Perkeretaapian Tahun 2020," The Directorate General of Railway, 2020.

[3] A. Sianipar, "Kajian penerapan teknologi pintu dengan pagar otomatis dan yellow box di perlintasan sebidang," *J. Penelit. Transp. Darat.*, vol. 22, no. 1, pp. 91–102, Jun. 2020, doi: 10.25104/jptd.v22i1.1603.

[4] S.H. Supangkat, A.A. Arman, R.A. Nugraha, and Y.A. Fatimah, "The implementation of Garuda Smart City Framework for smart city readiness mapping in Indonesia," *J. Asia-Pac. Stud.*, vol. 32, pp. 169–176, Mar. 2018, doi: 10.57278/wiapstokyu.32.0_169.

[5] V. Tsakanikas and T. Dagiuklas, "Video surveillance systems-current status and future trends," *Comput. Elect. Eng.*, vol. 70, pp. 736–753, Aug. 2018, doi: 10.1016/j.compeleceng.2017.11.011.

[6] R.P. Pratama and S.H. Supangkat, "Smart video surveillance system for level crossing: A systematic literature review," *2021 Int. Conf. ICT Smart Soc. (ICISS)*, 2021, pp. 1–5, doi: 10.1109/ICISS53185.2021.9533222.

[7] M.H. Kolekar, *Intelligent Video Surveillance Systems An Algorithmic Approach*. New York, NY, USA: CRC Press, 2017, doi: 10.1201/9781315153865.

[8] A. Hampapur *et al.*, "Smart video surveillance: Exploring the concept of multiscale spatiotemporal tracking," *IEEE Signal Process. Mag.*, vol. 22, no. 2, pp. 38–51, Mar. 2005, doi: 10.1109/MSP.2005.1406476.

[9] G.F. Shidik *et al.*, "A systematic review of intelligence video surveillance: Trends, techniques, frameworks, and datasets," *IEEE Access*, vol. 7, pp. 170457–170473, Nov. 2019, doi: 10.1109/ACCESS.2019.2955387.

[10] H. Sun, Y. Yu, K. Sha, and B. Lou, "mVideo: Edge computing based mobile video processing systems," *IEEE Access*, vol. 8, pp. 11615–11623, Dec. 2020, doi: 10.1109/ACCESS.2019.2963159.

[11] W. Yu *et al.*, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, Nov. 2018, doi: 10.1109/ACCESS.2017.2778504.

[12] W. Shi *et al.*, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[13] F. Wang *et al.*, "Deep learning for edge computing applications: A state-of-the-art survey," *IEEE Access*, vol. 8, pp. 58322–58336, Mar. 2020, doi: 10.1109/ACCESS.2020.2982411.

[14] D.R. Patrikar and M.R. Parate, "Anomaly detection using edge computing in video surveillance system: Review," *Int. J. Multimed. Inf. Retr.*, vol. 11, no. 2, pp. 85–110, Jun. 2022, doi: 10.1007/s13735-022-00227-8.

[15] A.C. Cob-Parro *et al.*, "Smart video surveillance system based on edge computing," *Sensors*, vol. 21, no. 9, pp. 1–20, May 2021, doi: 10.3390/s21092958.

[16] A. Gupta and P. Prabhat, "Towards a resource efficient and privacy-preserving framework for campus-wide video analytics-based applications," *Complex Intell. Syst.*, vol. 9, no. 1, pp. 161–176, Feb. 2023, doi: 10.1007/s40747-022-00783-w.

[17] M. Aazam, S. Zeadally, and K.A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Gener. Comput. Syst.*, vol. 87, pp. 278–289, Oct. 2018, doi: 10.1016/j.future.2018.04.057.

[18] A. Zaman, B. Ren, and X. Liu, "Artificial intelligence-aided automated detection of railroad trespassing," *Transp. Res. Rec., J. Transp. Res. Board,*, vol. 2673, no. 7, pp. 25–37, Jul. 2019, doi: 10.1177/0361198119846468.

[19] M.A.B. Fayyaz and C. Johnson, "Object detection at level crossing using deep learning," *Micromachines*, vol. 11, no. 12, pp. 1–16, Dec. 2020, doi: 10.3390/mi11121055.

[20] P. Sikora *et al.*, "Artificial intelligence-based surveillance system for railway crossing traffic," *IEEE Sens. J.*, vol. 21, no. 14, pp. 15515–15526, Jul. 2021, doi: 10.1109/jsen.2020.3031861.

[21] C. Sun *et al.*, "MCA-YOLOV5-light: A faster, stronger and lighter algorithm for helmet-wearing detection," *Appl. Sci.*, vol. 12, no. 19, pp. 1-19, Oct. 2022, doi: 10.3390/app12199697.

[22] X. Xu, X. Zhang, and T. Zhang, "Lite-YOLOv5: A lightweight deep learning detector for on-board ship detection in large-scene Sentinel-1 SAR images," *Remote Sens.*, vol. 14, no. 4, pp. 1–27, Feb. 2022, doi: 10.3390/rs14041018.

[23] M. Ali *et al.*, "RES: Real-time video stream analytics using edge enhanced clouds," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 792–804, Apr.–Jun. 2022, doi: 10.1109/TCC.2020.2991748.

[24] X. Xia *et al.*, "Cost-effective app data distribution in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 31–44, Jan. 2021, doi: 10.1109/TPDS.2020.3010521.

[25] H. Alawad, S. Kaewunruen, and M. An, "Learning from accidents: Machine learning for safety at railway stations," *IEEE Access*, vol. 8, pp. 633–648, Dec. 2020, doi: 10.1109/ACCESS.2019.2962072.

[26] U. Nepal and H. Eslamiat, "Comparing YOLOv3, YOLOv4 and YOLOv5 for autonomous landing spot detection in faulty UAVs," *Sensors*, vol. 22, no. 2, pp. 1–15, Jan. 2022, doi: 10.3390/s22020464.

[27] P. Sikora, M. Kiac, and M.K. Dutta, "Classification of railway level crossing barrier and light signalling system using YOLOv3," *2020 43rd Int. Conf. Telecommun. Signal Process. (TSP)*, 2020, pp. 528–532, doi: 10.1109/TSP49548.2020.9163535.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[29] J. Solawetz (2020) "What is Mean Average Precision (mAP) in Object Detection?," [Online], https://blog.roboflow.com/mean-average-precision/, access date: 15-Jul-2023.

[30] M. Veeramanikandan and S. Sankaranarayanan, "Publish/subscribe based multi-tier edge computational model in internet of things for latency reduction," *J. Parallel Distrib. Comput.*, vol. 127, pp. 18–27, May 2019, doi: 10.1016/j.jpdc.2019.01.004.

[31] D.J. Shin and J.J. Kim, "A deep learning framework performance evaluation to use YOLO in Nvidia Jetson platform," *Appl. Sci.*, vol. 12, no. 8, pp. 1–19, Apr. 2022, doi: 10.3390/app12083734.

[32] I. Resmadi, "Kajian moralitas teknologi pintu perlintasan kereta api (Studi kasus: Pintu perlintasan kereta api Cikudapateuh Bandung)," *J. Sosioteknol.* vol. 13, no. 2, pp. 84–90, Aug. 2014, doi: 10.5614/sostek.itbj.2014.13.2.2.

[33] S. Valladares *et al.*, "Performance evaluation of the Nvidia Jetson Nano through a real-time machine learning application," *Int. Conf. Intell. Hum. Syst. Integr.*, 2021, pp. 343–349, doi: 10.1007/978-3-030-68017-6_51.

[34] A. Al-Qamash, I. Soliman, R. Abulibdeh, and M. Saleh, "Cloud, fog, and edge computing: A software engineering perspective," *2018 Int. Conf. Comput. Appl. (ICCA)*, 2018, pp. 276–284, doi: 10.1109/COMAPP.2018.8460443.