Cloud-Based Architecture for YOLOv3 Object Detector using gRPC and Protobuf

Eko Rudiawan Jamzuri[1*], Hanjaya Mandala[2], Riska Analia[1], and Susanto[1]

[1]*Department of Electrical Engineering, Politeknik Negeri Batam*
Jl. Ahmad Yani, Tlk. Tering, Kec. Batam Kota, Kota Batam, Kepulauan Riau, 29461, Indonesia

[2]*Department of Electrical Engineering, National Taiwan Normal University*
162, Section 1, Heping E. Rd., Taipei City, 106, Taiwan

[*]*Corresponding author. Email: ekorudiawan@polibatam.ac.id*

Abstract— The deep learning-based object detector accuracy has surpassed conventional detection methods. Although implementation is still limited to hardware capabilities, this problem can be overcome by combining edge devices with cloud computing. The recent study of cloud-based object detector architecture is generally based on representational state transfer (RESTful web services), which uses a pooling system method for data exchange. As a result, this system leads to a low detection speed and cannot support real-time data streaming. Therefore, this study aims to enhance the detection speed in cloud-based object recognition systems using gRPC and Protobuf to support real-time detection. The proposed architecture was deployed on the Virtual Machine Instance (VMI) equipped with a Graphics Processing Unit (GPU). The gRPC server and YOLOv3 deep learning object detector were executed on the cloud server to handle detection requests from edge devices. Furthermore, the captured images from the edge devices were encoded into Protobuf format to reduce the message size delivered to the cloud server. The results showed that the proposed architecture improved detection speed performance on the client-side in the range of 0.27 FPS to 1.72 FPS compared to the state-of-the-art method. It was also observed that it could support multiple edge devices connection with slight performance degradation in the range of 1.78 FPS to 1.83 FPS, depending on the network interface used.

Keywords— cloud computing; gRPC; object detection; Protobuf; YOLO

I. Introduction

One of the sub-fields of computer vision is object detection, which interprets and understands an image by correctly estimating the object's location and object classes in the frame [1]. This object detection task is often needed in daily life applications, such as surveillance, military, transportation, and medical [2]. The accuracy of deep learning detectors has recently surpassed conventional methods that rely on informative region selection, feature extraction, and classification. This condition is why numerous deep learning architectures have been introduced recently. Some popular detector architectures which the researcher introduced are: You Only Look Once (YOLO) [3], Region-based Convolutional Neural Network (RCNN) [4], Single-Shoot Multibox Detector (SSD) [5], RetinaNet [6], and EfficientDet [7]. Although most deep learning models have surprising accuracy, most models have high computational costs that are inappropriate for low-power or slow computers. In contrast, object detection algorithms are generally needed to develop applications on cellular devices, embedded systems, and robotics with limited computational power and resources [8]. Merging edge devices and cloud computing can address hardware capability barriers in deep learning. A cloud server with scalable and high-performance processing capability can manage the inference by receiving an image from the edge devices. The most frequently used method is the REpresentational State Transfer Application Programming Interface (REST API) offered by third-party services.

Several studies have been conducted on these third-party object detection APIs. An example is the accuracy investigation of two commercial services, such as Microsoft Cognitive Services and Google Cloud Vision [9]. The results showed that both cloud services delivered a significant and acceptable accuracy level for running a helper application for blind people. Moreover, the accuracy comparison of the Tesseract Optical Character Recognition (OCR) and Google Cloud Vision for recognizing Thai vehicle registration certificates was investigated in [10]. It was discovered that Google Cloud Vision and Tesseract OCR achieved accuracies of 84.43% and 47.02%, respectively. Similarly, Google Cloud Vision was proposed to build a vision system for people with no medical expertise [11]. The experimental results revealed a promising performance of the proposed technology as it can provide pertinent information about the given image. Finally, [12] presented that the Google AutoML service can automatically explore and train the model on the cloud. The user only needs to provide the image dataset, while the service automatically finds, trains, and infers the best model for the particular case. This experiment reported that Google AutoML attained an average accuracy of 91.6% during the model evaluation.

Another study by [13] employed the Azure Machine Learning service to recognize the frailty and senility conditions of the elderly. The gait sequences were acquired through smartphones, and the Spatio-Temporal was analyzed to obtain the features. Afterward, these features are uploaded to the cloud as input data, and then the cloud generates a diagnostic result. Furthermore, [14] introduced an attendance system using a hybrid cloud-edge detection based on YOLOv3 for facial detection and the Microsoft Azure Face API for recognizing the human face in a database. The camera in the classroom

Received 17 May 2022, Accepted 15 July 2022, Published 20 July 2022.
DOI: https://doi.org/10.15294/jte.v14i1.36537

photographed students twice throughout the period, at the beginning and end, to ensure they attended the entire session.

Some studies have developed a custom architecture for minimizing services cost using REST API that runs on a cloud server to provide service for edge devices. Examples include Transport for London (TfL) in [15], which utilized the YOLOv3 for the JamCam that is implementable on any computer with NVIDIA GPU. Furthermore, [16] presented a way of hosting a computer vision API on Amazon Web Services to recognize and classify multiple items in an image. The API accepted the user-supplied image and returned the object containing the most abundant color. Finally, [17] proposed a cloud-edge collaborative system using a YOLOv3 object detector for an electronic gastroscopy system, and the framework achieved real-time performance for disease screening.

The third-party cloud services' accuracy and precision for specific problems have been investigated in [9]–[14] regarding cloud edge collaborative object detection. However, studies on detection speed performance are limited, particularly on custom cloud-edge architectures. It has been observed that the proposed custom architecture by [17] still results in slow detection speed and is designed for a single client. Therefore, this study aims to improve the detection speed by proposing gRPC frameworks and Protobuf for cloud-edge communication schemes. The architecture is designed to handle multiple edge devices connection simultaneously with little performance degradation. The gRPC frameworks and Protobuf were demonstrated to handle multiple edge-device requests. Moreover, the computational speed of each sub-process was examined to discover its significant improvement.

The remaining part of this article is arranged as follows. Section II discusses the study approach, which includes the object detection model and communication scheme between the cloud server and edge devices. Section III elaborates the proposed architecture's results by employing Raspberry Pi as an edge device. Finally, Section IV contains the summary of the results.

II. Method

The proposed architecture was designed by selecting a cloud server capable of implementing a deep learning-based object detector, considering the hardware capability. Furthermore, the YOLOv3 object detector was implemented on the cloud server, and optimized pre-trained weight was utilized to accelerate detection speed. Finally, a communication mechanism was established between the client and server using Remote Procedure Call (RPC), while images were compressed and encoded into serial data structures for the messaging system to accelerate data transmission. The following sub-sections describe the method employed in-depth.

A. Cloud-Based Architecture

Figure 1 shows the proposed cloud-based architecture consisting of a cloud server and edge devices. A Google Virtual Machine Instance (VMI) was used as the cloud server in Changhua City, Taiwan, and it was equipped with GPU NVIDIA Tesla K80, with 2.91 teraflops double-precision performance. It also consists of 2×virtual CPUs, 7.5 GB RAM, and 50 GB storage. Furthermore, a containerized application was delivered using Docker to ensure a simplified deployment, maintenance, and migration process. The program on the cloud server comprised a gRPC server, Protobuf Decoder, Protobuf Encoder, Image Preprocessing, and YOLOv3. The primary component of a deep learning object detector is YOLOv3, and

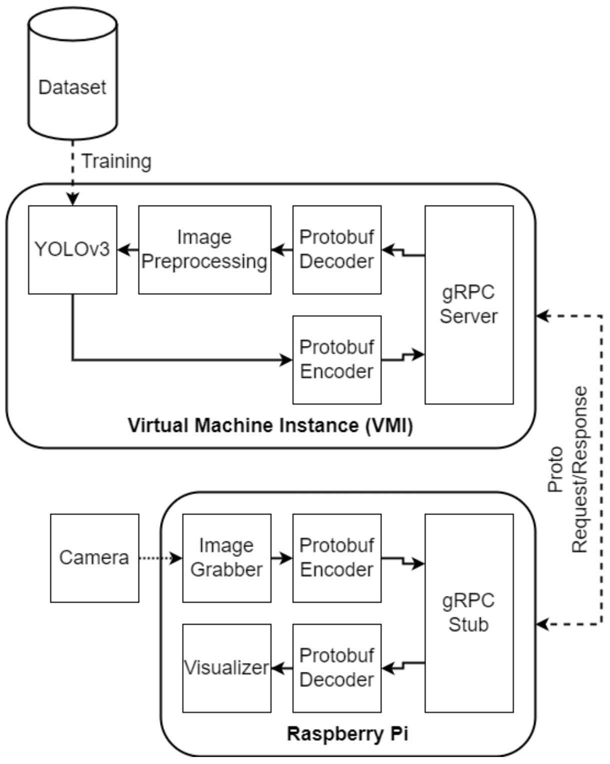


Figure 1. Proposed cloud-based architecture

the gRPC server manages data communication while the decoder/encoder translates the data format. Image preprocessing was used to preprocess image data before inference.

The Raspberry Pi 3, a low-power embedded computer with a 1.2 GHz quad-core processor, was utilized as the edge device. The web camera was attached to the Raspberry Pi as a vision sensor, and the Ethernet or WiFi connection connects the edge device to the cloud server. Figure 1 shows the sub-process in the edge client. The edge client program utilized the gRPC Stub for communicating with the server. Meanwhile, there were sub-processes on the edge device for capturing, displaying, and encoding/decoding images to the Protobuf format. The Client-Server Communication section covers a detailed client-server communication scheme design.

B. Deep Learning Object Detector

The approach for recognizing objects was an optimized YOLOv3, which is an improved version of the YOLO base model [3] and YOLO9000 [18]. In order to detect objects, YOLOv3 takes the 416×416 pixels image and extracts image features using the Darknet-53 feature extractor. The Darknet-53 feature extractor has a thicker layer, enabling it to achieve more accurate detection than the YOLO base model. Furthermore, due to the single-stage approach in YOLOv3, its detection time is quicker than other deep learning models.

The YOLOv3 recognition process begins by partitioning the input image into $n \times n$ cell patch and assigning each ground truth object with the bounding box anchor. Furthermore, the output of the Darknet-53 feature extractor was used for predicting object classes denoted as t_o and four box parameters, represented by t_x, t_y, t_w, t_h. The YOLOv3 bounding box is composed of four coordinate pairs, such as the bounding box's center point (b_x, b_y), the bounding box's width and height (b_w, b_h) [19], which were calculated from bounding box parameters by applying (1) - (4), and the object class was obtained from (5).

$$b_x = \sigma(t_x) + c_x \quad (1)$$

$$b_y = \sigma(t_y) + c_y \quad (2)$$

$$b_w = p_w e^{t_w} \quad (3)$$

$$b_h = p_h e^{t_h} \quad (4)$$

$$Pr(\text{object}) * IoU(\text{b}, \text{object}) = \sigma(t_o) \quad (5)$$

The YOLOv3 predicted parameters and the bounding box is seen in Figure 2, where the variable c_x and c_y are denoted as the offset cell from the image origin coordinate. Function $\sigma(t_x)$ and $\sigma(t_y)$ generated the center point coordinates, which were defined from the cell's origin. The dashed rectangle shown provided information about the prior bounding box prediction, having a width and height of p_w and p_h, respectively.

Several studies have been conducted to optimize deep learning performance. For instance, the reduction of superfluous channels in the neural network by using the combination of Parameter Pruning (PP) with Feature-map Quantization (FQ) and Parameter Quantization (PQ) approaches [20]. These combinations led to SE model generation that was condensed by only 9.76% [20]. A systematic approach was also proposed for converting Tiny YOLOv2 floating-point weight to 8-bit fixed-point representation [21]. The results showed a significant reduction in hardware consumption with only 0.3% inaccuracy. This present study optimized the YOLOv3 by quantizing weights into a 16-bit floating-point representation, and the approach is similar to [22]–[25], which used NVIDIA TensorRT Optimizer for transforming and quantifying weights. In contrast with [22]–[25], which used the compact model in embedded GPU, this study was implemented on GPU located on the cloud server.

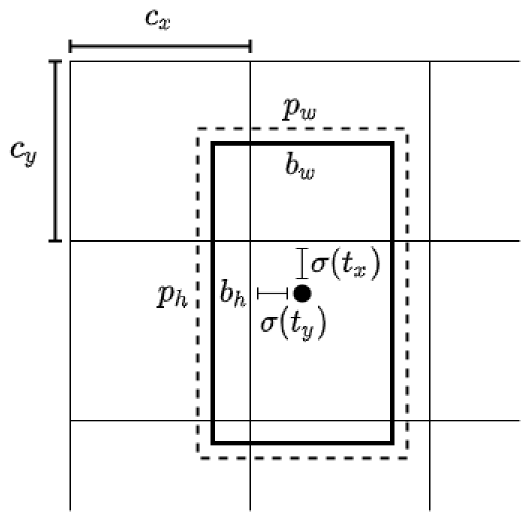
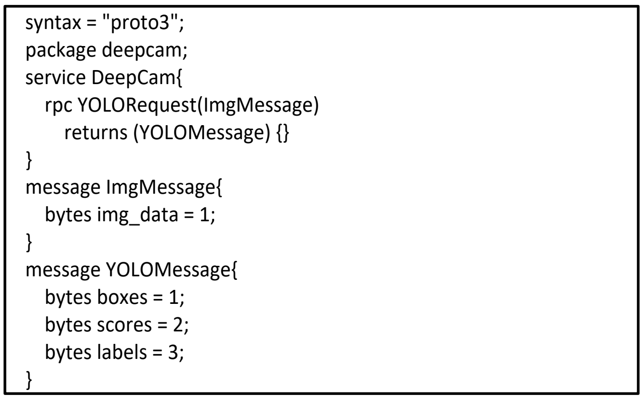


Figure 2. YOLOv3 bounding box prediction



```
syntax = "proto3";
package deepcam;
service DeepCam{
   rpc YOLORequest(ImgMessage)
      returns (YOLOMessage) {}
}
message ImgMessage{
   bytes img_data = 1;
}
message YOLOMessage{
   bytes boxes = 1;
   bytes scores = 2;
   bytes labels = 3;
}
```

Figure 3. Protobuf proto file

C. Client-Server Communication

This study focused on the client and server data exchange, including encoding/decoding image data. We utilized gRPC, a Google open-source request-response protocol framework that is cross-platform and language-independent. This RPC was used as an executor subroutines on several machines connected over a shared network [26]. Moreover, the gRPC was designed based on an HTTP/2 transport protocol, enabling bi-directional real-time data streaming. It was observed that sending uncompressed raw images was an inefficient communication as it potentially increased latency. Therefore, a defined compression format and communication protocol were required to facilitate the data transmission.

It is important to note that JPEG was used to compress and reduce image data size and encode it into Protocol Buffer (Protobuf) structured data. Afterward, the gRPC Stub on the edge device requests service by sending this encoded data. The selection of Protobuf was based on a recent trade-off performed by [26], showing that Protobuf had a lower network load than JavaScript Object Notation (JSON) or Binary Javascript Object Notation (BSON) format. Moreover, the examination by [26] of the application layer's performance, messaging protocols, and binary serialization formats, concluded that although Protobuf's competitor, Constrained Application Protocol (CoAP), offers the lowest latency and overhead, it cannot ensure reliable transmission. Meanwhile, Protobuf supported a faster serialization and three-fold reduction in serialized messages compared to other serialization libraries like Flatbuffers [27].

Figure 3 shows a proto file constructed in the Protobuf format, comprising a single service and two message protocols for handling client-server communication. The DeepCam service includes a YOLORequest function, which inputs an ImgMessage and returns a YOLOMessage for processing edge-devices requests. It is important to note that the ImgMessage and YOLOMessage were data exchange messaging protocols. The first helps to transmit encoded JPG image data from edge devices to the cloud server, while the second communicates detection results from the server to edge devices. YOLOMessage consists of structured data in boxes, scores, and labels that specify an object's Region of Interest (ROI), confidence score, and class name.

Figure 4 shows the cloud server and edge devices' flowcharts. In Figure 4 (a), the cloud server first opens a dedicated port for gRPC communication while the server waits for a detection request from the edge device. When there is a detection request, the message has to be decoded into a raw image. The decoding converts the image into a Python Imaging Library (PIL) format, which is further transformed into a fixed square size of 416×416 pixels, and its values were normalized to the floating-point numbers 0-1 to match the input format for the YOLOv3. Furthermore, the cloud server executed the detection task on GPU to identify images' objects for producing boxes, scores, and labels. These boxes contained information about an object's bounding, while the scores represented the degree of confidence in an identified object, and the labels denoted an index to a particular label in the list of objects. Subsequently, this data was transformed into a byte array compatible with the Protobuf standard, and the cloud server sent the detection result via gRPC in Protobuf format.

Figure 4 (b) shows the process of the client-side flowchart algorithm in which the edge device first established a gRPC connection with the cloud server. When the connection is successful, the edge device grabs an image from the camera

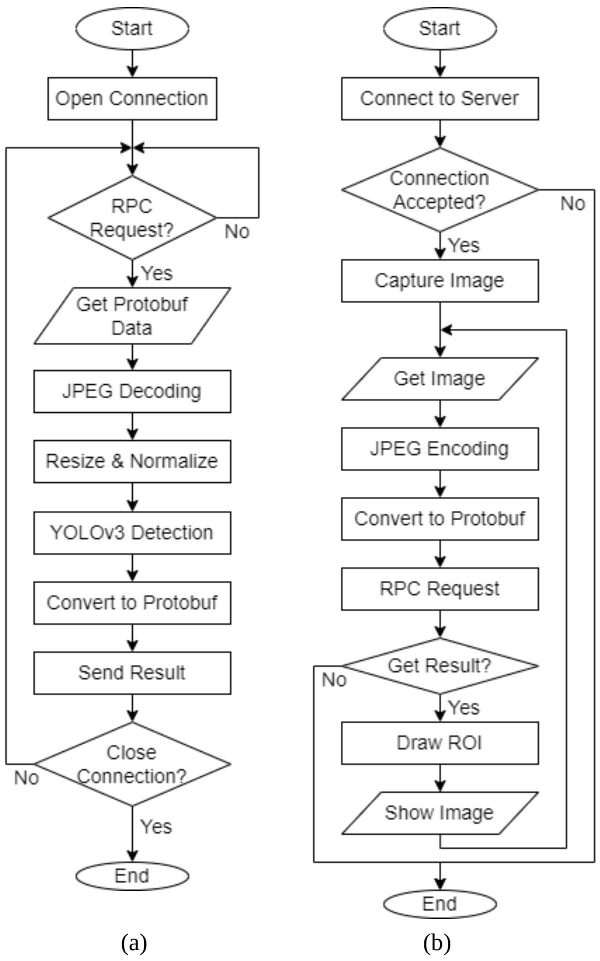


Figure 4. Flowchart on (a) cloud server and (b) edge devices

sensor and then resizes and encodes it into JPG format. Afterward, the JPG image is encoded into a YOLOMessage byte array known as Protobuf data. The edge device then invoked the DeepCam service to request a detection task, and the cloud server responds with a detection result message containing boxes, scores, and labels for the identified object. Finally, these boxes, scores, and labels returned from the server were marked using the OpenCV library to indicate the recognized object in the raw image.

III. Results and Discussion

Several experiments have been conducted to evaluate the performance of the proposed approach. The first experiment was performed to determine the object detection's success, followed by the analysis of its speed on the client-side, conducted with one and two clients. The computation time for each sub-process was also measured, and the data obtained were compared with the cutting-edge techniques to identify the significant improvement achieved. The details of the obtained results are explained in the remaining section.

A. Object Detection Result

The proposed method was evaluated using an image captured from the edge device camera sensor. According to Figure 5, the cloud-based object detector successfully recognized the person and sports ball. The visualization on the edge devices showed three detected objects with a confidence score above 0.9. Furthermore, all detected objects have correct classes, and the system correctly marks their location. The detection speed was also visualized at the top left of the image, which described the total number of frames processed in one second. Figure 5 shows that the achieved detection speed for an

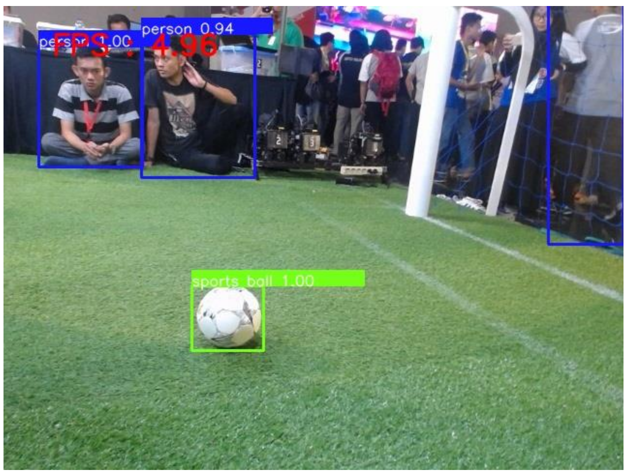


Figure 5. Visualization of object detection results on edge devices

image containing a sports ball and persons is 4.96 FPS. Our detection rate is slower than [25], which presented YOLOv4 and TensorRT optimization on the embedded platform. They could attain detection rates higher, between 31.10 and 35.81 FPS. Our system's slower detection rate is due to the latency of communication requests from the edge to the server. We will elaborate on this observation in the following sub-section.

B. Detection on Single Edge Device

The performance of the cloud object detection system in handling a proto-request from the single edge device was examined. First, we compared ethernet and WiFi interfaces to study the detection speed performance presented in Frame Per Seconds (FPS). It is important to note that the sub-processes in the system, which include capturing, resizing, encoding, and requesting detection results, were analyzed to evaluate the time-critical process. It was observed that the measured internet speed via ethernet interface has a download speed of 49.57 MBps and an upload speed of 83.57 MBps.

Table I describes the test results elaborated on ethernet interface with different image resolution inputs. It was observed that the maximum detection speed reached approximately 6.28 FPS with a 640×480 pixels image captured by the camera. The larger image resolution reduced the detection speed because the larger ImgMessage packet size had to be sent to the cloud server. Meanwhile, the most time-consuming process was requesting the detection results from the cloud server, which took 127.94 ms to 178.29 ms. The second slowest sub-process was encoding the raw image data, which takes about 20.31ms to 141.40ms. This result means that image processing is associated with its resolution.

Table II shows the system's performance in implementing the proposed method via WiFi interface. It was observed that the internet connection speed was 10.90 MBps and 15.31 MBps for download and upload, respectively. As a result, the maximum detection speed was achieved at 4.83 FPS with an image resolution of 800×448 pixels. Unlike the Ethernet connection, the detection speed of bigger images was sometimes faster compared to the small. For example, the detection speed with the resolution of 640×480 pixels achieved 4.70 FPS, but when increased to 800×448 pixels, the detection speed rose to 4.83 FPS. This phenomenon is often affected by the unstable WiFi connection during the experiment. This phenomenon was also reported by [27] while analyzing the latency of several protocols to transmit serialization data. It was therefore concluded that the time required by the process to request the detection results from the cloud server was 161.94 ms.

TABLE I. DETECTION SPEED OF SINGLE EDGE DEVICE USING ETHERNET INTERFACE

Image Resolution	Capture	Resize	Encode	Request	Draw	Detection Speed
640×480 pixels	4.81 us	3.32 ms	20.31 ms	127.94 ms	1.68 ms	6.28 FPS
800×448 pixels	3.89 us	11.01 ms	27.38 ms	135.55 ms	2.09 ms	5.54 FPS
800×600 pixels	7.01 us	12.57 ms	30.69 ms	138.00 ms	2.01 ms	5.21 FPS
848×480 pixels	5.26 us	9.11 ms	31.31 ms	136.17 ms	1.69 ms	5.45 FPS
960×540 pixels	4.30 us	12.25 ms	38.93 ms	138.16 ms	1.71 ms	5.13 FPS
1024×576 pixels	7.35 us	15.26 ms	48.45 ms	140.35 ms	2.04 ms	4.67 FPS
1280×720 pixels	7.46 us	21.45 ms	60.13 ms	148.56 ms	2.14 ms	4.16 FPS
1600×896 pixels	7.61 us	28.58 ms	99.02 ms	187.71 ms	1.94 ms	3.08 FPS
1920×1080 pixels	5.08 us	24.89 ms	141.40 ms	178.29 ms	1.13 ms	2.86 FPS

TABLE II. DETECTION SPEED OF SINGLE EDGE DEVICE USING WIFI INTERFACE

Image Resolution	Capture	Resize	Encode	Request	Draw	Detection Speed
640×480 pixels	4.21 us	2.33 ms	24.01 ms	179.41 ms	0.02 ms	4.70 FPS
800×448 pixels	5.45 us	11.31 ms	28.37 ms	161.94 ms	0.02 ms	4.83 FPS
800×600 pixels	6.71 us	10.27 ms	32.78 ms	178.85 ms	0.04 ms	4.39 FPS
848×480 pixels	5.48 us	16.31 ms	37.69 ms	186.94 ms	0.02 ms	4.06 FPS
960×540 pixels	4.40 us	10.94 ms	2.89 ms	174.02 ms	0.01 ms	4.48 FPS
1024×576 pixels	8.62 us	18.46 ms	48.04 ms	205.74 ms	0.02 ms	3.56 FPS
1280×720 pixels	5.15 us	15.63 ms	72.71 ms	213.98 ms	0.02 ms	3.27 FPS
1600×896 pixels	5.06 us	20.07 ms	87.16 ms	271.21 ms	0.02 ms	2.62 FPS
1920×1080 pixels	7.33 us	27.07 ms	120.64 ms	823.51 ms	0.02 ms	1.17 FPS

TABLE III. DETECTION SPEED OF MULTIPLE EDGE DEVICES USING ETHERNET AND WIFI INTERFACE

Resolution	Detection Speed using Ethernet Interface				Detection Speed using WiFi Interface			
	Device 1	Device 2	Average	Differences	Device 1	Device 2	Average	Differences
640×480 pixels	4.46 FPS	4.45 FPS	4.46 FPS	0.01 FPS	3.66 FPS	4.59 FPS	4.13 FPS	0.93 FPS
800×448 pixels	4.10 FPS	4.13 FPS	4.12 FPS	0.03 FPS	2.89 FPS	4.07 FPS	3.48 FPS	1.18 FPS
800×600 pixels	4.14 FPS	4.12 FPS	4.13 FPS	0.02 FPS	3.32 FPS	4.24 FPS	3.78 FPS	0.92 FPS
848×480 pixels	4.05 FPS	3.99 FPS	4.02 FPS	0.06 FPS	2.82 FPS	4.04 FPS	3.43 FPS	1.22 FPS
960×540 pixels	4.11 FPS	3.57 FPS	3.84 FPS	0.54 FPS	1.30 FPS	4.10 FPS	2.70 FPS	2.80 FPS
1024×576 pixels	3.23 FPS	3.42 FPS	3.33 FPS	0.19 FPS	1.17 FPS	3.10 FPS	2.14 FPS	1.93 FPS
1280×720 pixels	3.75 FPS	3.49 FPS	3.62 FPS	0.26 FPS	2.29 FPS	3.12 FPS	2.71 FPS	0.83 FPS
1600×896 pixels	3.85 FPS	3.34 FPS	3.60 FPS	0.51 FPS	0.84 FPS	2.45 FPS	1.65 FPS	1.61 FPS
1920×1080 pixels	2.47 FPS	2.42 FPS	2.45 FPS	0.05 FPS	0.70 FPS	1.50 FPS	1.10 FPS	0.80 FPS

C. Detection of Multiple Edge Devices

This section illustrates the performance evaluation of multiple edge device connections. Specifically, two edge devices were connected to the same router via Ethernet or WiFi. Before the experiment, both devices' download and upload speeds were measured. It was observed that during the Ethernet connection usage, the first edge device supported about 49.57 and 83.57 MBps download and upload rates, while that of the second edge device had 55.16 and 92.45 MBps, respectively. These speeds slightly decreased when the devices were connected through WiFi. For example, the internet speed in the first edge device was 10.90 MBps and 15.31 for downloading and uploading, respectively, while in the second edge device, it was 13.71 MBps and 16.84, respectively. The detection speed reduction when the cloud server simultaneously handled the proto-request from multiple edge devices was further evaluated.

Table III summarizes the implementation results of multiple edge connections on the Ethernet and WiFi interface. As described in Table III, the detection speed discrepancies ranged from 0.01 FPS to 0.54 FPS while both devices connected through ethernet. The highest speed disparities occurred when the 960×540 pixels image resolution was utilized in edge devices. The maximum degradation detection speed of 1.83 FPS was observed when detecting the 640×480 pixels image compared to the single device connection. Moreover, the drop detection speed fell when the system recognized a higher resolution image. For example, the FPS loss was only 0.42 in the 1920×1080 pixels image.

The evaluation of using WiFi interface on both edge devices described that the differences in detection speed range were around 0.80 FPS to 2.80 FPS. It was observed that the highest differences were recorded when 960×540 pixels image was employed, while the lowest was obtained during the utilization of 1920×1080 pixels. Furthermore, the average multiple client detection speed on the WiFi interface varied between 0.07 FPS to 1.78 FPS compared to the single-client performance. In this scenario, the highest differences occurred when the 960×540 pixels image was used, and the maximum degradation reached 1.78 FPS compared to the single client detection speed.

D. Comparison with State-of-the-art Method

The detection speed of the proposed architecture was compared with the study in [17] to prove its significant improvement. We selected [17] because its approach utilized a similar edge-cloud collaboration and the same YOLOv3 model as the detector. However, it did not describe the image compression, encoding process, and model optimization but directly processed the raw image from the video.

Table IV describes comparison results with the SOTA method. By head-to-head comparison using the same YOLOv3, our architecture improves detection speed by about 1.72 FPS while using the ethernet connection. Even using a WiFi connection, our method still resulting a faster processing time of about 0.27 FPS. Experimental results from [17] showed that the variation of the λ value slightly improves detection speed. However, it still cannot beyond our cloud architecture performance.

TABLE IV. COMPARISON WITH STATE-OF-THE-ART METHOD

Method	Detector Model	Network Interface	Maximum Detection Speed
[17]	**YOLOv3**	**Unknown**	**4.56 FPS**
	YOLOv3($\lambda = 1.5$)	Unknown	4.62 FPS
	YOLOv3($\lambda = 2.0$)	Unknown	4.59 FPS
	YOLOv3($\lambda = 2.5$)	Unknown	4.58 FPS
Ours	**YOLOv3**	**Ethernet**	**6.28 FPS**
		WiFi	**4.83 FPS**

IV. CONCLUSION

The cloud architecture for object detection based on deep learning has been described. A communication mechanism was developed for real-time detection on multiple edge devices using gRPC and Protobuf. Experimentally, the object detection successfully runs on the proposed architecture. It was observed that the maximum degradation performance when handling multiple requests ranges from 1.78 FPS to 1.83 FPS. Compared to the state-of-the-art, the proposed architecture improved detection speed performance from 0.27 FPS to 1.72 FPS, depending on the network interface. The result of time consumption in each sub-process revealed that the highest time consumed was recorded when requesting data from the cloud due to the data payload. Further studies need to investigate other binary data-interchange formats to examine decreased latency affecting detection speed escalation. Several potential improvements are achievable by using a recent lightweight object detection model.

ACKNOWLEDGMENT

This work was completed as part of the final project for the course Application of Wireless Networking Systems in the Internet of Things (ABC8420), taught by the Department of Electronics and Computer Engineering at the National Taiwan University of Science and Technology. We want to express our gratitude to the National Taiwan University of Science and Technology (NTUST), National Taiwan Normal University (NTNU), and Politeknik Negeri Batam (Polibatam) for providing the facilities necessary to perform this research.

REFERENCES

[1] Z.-Q. Zhao, P. Zheng, S. Xu, and X. Wu, "Object Detection With Deep Learning: A Review," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.

[2] L. Aziz, M. S. Bin Haji Salam, U. U. Sheikh, and S. Ayub, "Exploring Deep Learning-Based Architecture, Strategies, Applications and Current Trends in Generic Object Detection: A Comprehensive Review," *IEEE Access*, vol. 8, pp. 170461–170495, 2020.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 1, pp. 142–158, Jan. 2016.

[5] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9905 LNCS, Springer Verlag, 2016, pp. 21–37.

[6] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, Feb. 2020.

[7] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 10778–10787.

[8] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang, "Towards Real-Time Object Detection on Embedded Systems," *IEEE Trans. Emerg. Top. Comput.*, vol. 6, no. 3, pp. 417–431, 2018.

[9] D. Paulino, A. Reis, H. Paredes, H. Fernandes, and J. Barroso, "Usage of Artificial Vision Cloud Services as Building Blocks for Blind People Assistive Systems," *Int. J. Recent Technol. Eng.*, vol. 8, no. 2S10, pp. 453–458, Oct. 2019.

[10] K. Thammarak, P. Kongkla, Y. Sirisathitkul, and S. Intakosum, "Comparative analysis of Tesseract and Google Cloud Vision for Thai vehicle registration certificate," *Int. J. Electr. Comput. Eng.*, vol. 12, no. 2, p. 1849, Apr. 2022.

[11] I. K. G. Darma Putra, D. M. Sri Asra, I. G. N. Dwiva Hardijaya, I. G. G. Surya Prabawa, and I. M. A. Satia Widiatmika, "Medical vision: web and mobile medical image retrieval system based on google cloud vision," *Int. J. Electr. Comput. Eng.*, vol. 10, no. 6, p. 5974, Dec. 2020.

[12] Y. Zeng and J. Zhang, "A machine learning model for detecting invasive ductal carcinoma with Google Cloud AutoML Vision," *Comput. Biol. Med.*, vol. 122, p. 103861, Jul. 2020.

[13] M. Nieto-Hidalgo, F. J. Ferrández-Pastor, R. J. Valdivieso-Sarabia, J. Mora-Pascual, and J. M. García-Chamizo, "Gait Analysis Using Computer Vision Based on Cloud Platform and Mobile Device," *Mob. Inf. Syst.*, vol. 2018, pp. 1–10, 2018.

[14] S. Khan, A. Akram, and N. Usman, "Real Time Automatic Attendance System for Face Recognition Using Face API and OpenCV," *Wirel. Pers. Commun.*, vol. 113, no. 1, pp. 469–480, Jul. 2020.

[15] H. M. Gan, S. Fernando, and M. Molina-Solana, "Scalable object detection pipeline for traffic cameras: Application to Tfl JamCams," *Expert Syst. Appl.*, vol. 182, p. 115154, Nov. 2021.

[16] K. A. Jadhav, "Building and hosting a computer vision api on aws using an ec2 instance," *Int. J. Sci. Technol. Res.*, vol. 8, no. 12, pp. 857–862, 2019.

[17] S. Ding, L. Li, Z. Li, H. Wang, and Y. Zhang, "Smart electronic gastroscope system using a cloud–edge collaborative framework," *Futur. Gener. Comput. Syst.*, vol. 100, pp. 395–407, Nov. 2019.

[18] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7263–7271.

[19] H. Chen, Z. He, B. Shi, and T. Zhong, "Research on Recognition Method of Electrical Components Based on YOLO V3," *IEEE Access*, vol. 7, pp. 157818–157829, 2019.

[20] J.-Y. Wu, C. Yu, S.-W. Fu, C.-T. Liu, S.-Y. Chien, and Y. Tsao, "Increasing Compactness of Deep Learning Based Speech Enhancement Models With Parameter Pruning and Quantization Techniques," *IEEE Signal Process. Lett.*, vol. 26, no. 12, pp. 1887–1891, Dec. 2019.

[21] Y. J. Wai, Z. B. M. Yussof, and S. I. bin Md Salim, "Hardware Implementation and Quantization of Tiny-Yolo-v2 using Open CL," *Int. J. Recent Technol. Eng.*, vol. 8, no. 2S6, pp. 808–813, Sep. 2019.

[22] M. A. Farooq, W. Shariff, and P. Corcoran, "Evaluation of Thermal Imaging on Embedded GPU Platforms for Application in Vehicular Assistance Systems," *IEEE Trans. Intell. Veh.*, pp. 1–1, 2022.

[23] X. Li, B. He, K. Ding, W. Guo, B. Huang, and L. Wu, "Wide-Area and Real-Time Object Search System of UAV," *Remote Sens.*, vol. 14, no. 5, p. 1234, Mar. 2022.

[24] H. Feng, G. Mu, S. Zhong, P. Zhang, and T. Yuan, "Benchmark Analysis of YOLO Performance on Edge Intelligence Devices," *Cryptography*, vol. 6, no. 2, p. 16, Apr. 2022.

[25] 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, p. 3734, Apr. 2022.

[26] S. Kiraly and S. Szekely, "Analysing RPC and Testing the Performance of Solutions," *Informatica*, vol. 42, no. 4, pp. 555–561, Sep. 2018.

[27] D. P. Proos and N. Carlsson, "Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV," in *IFIP Networking 2020 Conference and Workshops, Networking 2020*, 2020, pp. 10–18. Accessed: Jul. 08, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9142787