applied sciences

Article

Inference Latency Prediction Approaches Using Statistical Information for Object Detection in Edge Computing

Gyuyeol Kong [1] and Yong-Geun Hong [2,*]

[1] Division of Mechanical and Electronics Engineering, Hansung University, Seoul 02876, Republic of Korea; gykong@hansung.ac.kr
[2] Department of Artificial Intelligence & Convergence, Daejeon University, Daejeon 34520, Republic of Korea
* Correspondence: yghong@dju.kr

Abstract: To seamlessly deliver artificial intelligence (AI) services using object detection, both inference latency from a system perspective as well as inference accuracy should be considered important. Although edge computing can be applied to efficiently operate these AI services by significantly reducing inference latency, deriving an optimized computational offloading policy for edge computing is a challenging problem. In this paper, we propose inference latency prediction approaches for determining the optimal offloading policy in edge computing. Since there is no correlation between the image size and inference latency during object detection, approaches to predict inference latency are required for finding the optimal offloading policy. The proposed approaches predict the inference latency between devices and object detection algorithms by using their statistical information on the inference latency. By exploiting the predicted inference latency, a client may efficiently determine whether to execute an object detection task locally or remotely. Through various experiments, the performances of predicted inference latency according to the object detection algorithms are compared and analyzed by considering two communication protocols in terms of the root mean square error. The simulation results show that the predicted inference latency matches the actual inference latency well.

Keywords: object detection; edge computing; offloading policy; inference latency; inference latency prediction

check for updates

Citation: Kong, G.; Hong, Y.-G. Inference Latency Prediction Approaches Using Statistical Information for Object Detection in Edge Computing. *Appl. Sci.* **2023**, *13*, 9222. https://doi.org/10.3390/app13169222

Academic Editor: Mirosław Klinkowski

Received: 12 July 2023
Revised: 29 July 2023
Accepted: 11 August 2023
Published: 14 August 2023

1. Introduction

As artificial intelligence (AI) technology develops and begins to be applied to various fields, various services using AI technology are being developed. Machine learning (ML), which is the core of AI technology, has been mostly researched in the direction of improving inference accuracy. AI models with high accuracy require vast amounts of data and parameters. OpenAI's generative pre-trained transformer 3 (GPT-3) model, widely used in the field of natural language processing, has used a total of 175 billion parameters and about 500 billion learning data to develop the model [1].

AI models should be provided to customers so that they can be applied in real life. To provide AI technology at a price such that it can be released as a service or product, technologies such as TensorFlow Serving [2], TorchServe [3], the Nvidia Transition Server [4], and Intel OpenVINO [5] have been studied in recent years to efficiently provide AI services focusing on inference rather than learning.

Edge computing has attracted a lot of attention because it can efficiently operate complex AI services such as object detection and mobile object tracking by offloading computing and processing data to a cloud and edge devices [6–10]. By applying edge computing to object detection, performance can be improved, which can be used to reduce inference latency and energy consumption and increase inference accuracy [9,10]. To maximize performance through edge computing, an offloading policy that determines which edge or cloud devices to execute the workload plays a very important role, and inference latency, communication latency, and energy consumption should be considered to derive an optimal offloading policy. However, since inference latency is affected by the characteristics of inference models, devices, and images, it is difficult to derive an offloading policy by reflecting the actual inference latency in practice. Therefore, it is required to derive an accurate offloading policy that takes into account the predicted inference latency.

In this paper, we introduce a resource optimization approach in terms of the inference latency of object detection in edge computing from a system perspective for an efficient AI service by using the TensorFlow serving system that is closely related to the ML model deployment of AI models. Inference latency is a very important factor to be considered in AI service systems, and finding optimal offloading policies in terms of inference latency when edge computing is considered is a very challenging problem. Information on the inference latency is essential to find the optimal offloading policy in the scheduling phase, and in many studies [11–21], inference latency has been set to be proportional to file size. However, in object detection, there is no correlation between the file size and inference latency, and the inference latency is determined by the characteristics of the image, requiring additional processes to calculate the inference latency. We propose two inference latency prediction approaches for object detection. The first method is aimed at inference latency prediction for the client, edge, and cloud devices, and the second method is focused on inference latency prediction for the object detection algorithms by using their statistical information on inference latency. By employing the predicted inference latency, the client determines whether to perform the object detection task locally or remotely on the edge and cloud for the optimal offloading policy. For the implementation of AI service systems, the region-based fully convolutional network (RFCN) [22] and single-shot multibox detector (SSD-MobileNet) [23] algorithms are considered as the two-stage and single-stage object detectors, respectively, and the representational state transfer (REST) [24,25] and Google remote procedure call (gRPC) [26] protocols are used as communication protocols. Through various experiments, the performances of predicted inference latency is compared and analyzed according to the object detection algorithms and communication protocols.

The main contributions of the paper are summarized as follows.

1. A system with AI technology was constructed using an object detection service as an AI service, and the performance of the object detection model on this system was studied from the perspective of the inference latency of the entire system, deviating from the inference accuracy;
2. When performing object detection, we reveal that there is no correlation between image size and inference latency, which presents a problem—that inference latency prediction is required for scheduling purposes;
3. We propose an approach for inference latency prediction during object detection in edge computing. The proposed approaches predict the inference latency between devices and object detection algorithms by using the statistical information on inference latency;
4. By exploiting the predicted inference latency, the client may effectively determine whether to perform the object detection task locally or remotely. By exploiting this scheduling phase, the overall performance of AI services could be improved;
5. Through simulation results, we demonstrate that the performances of the proposed approaches were compared and analyzed according to the object detection algorithms (RFCN and SSD-MobileNet) in consideration of two communication protocols (gRPC and REST).

The rest of the paper is organized as follows. Section 2 presents related works on AI services with the optimization of edge computing. In Section 3, we present the system model. Section 4 presents the proposed inference latency prediction approaches. Simulation results are presented in Section 5, and Section 6 concludes the paper.

2. Related Works

Recently, the operation of efficient AI services using edge computing in various applications has been studied. Depending on the nature of AI-based applications, edge computing could be used as a way to improve power consumption, inference accuracy, or inference delay performance, and it is important to consider what specific tasks to execute on which devices to maximize edge computing performance. Offloading strategies are essential for the effective use of edge computing, and various offloading strategies are being introduced in AI services.

Wu et al. [27] have studied the edge computing-driven AI service for object detection in low-light conditions. Since low-light conditions could reduce object detection performance, they proposed an end-to-end methodology for image enhancement and object detection. Their proposed structure consists of an image enhancement stage operating on cloud and an edge-based object detection stage performing in edge. In the first stage, the enhancement neural network computes the enhanced illumination portion of a low-light image. Then, edge devices can perform object detection accurately and quickly based on cloud computing information feature maps. By using this structure, the detection performance is significantly improved in low-light conditions.

Wu et al. [28] have developed an edge computing-based mobile object tracking method in the internet of things. For IoT devices, it is very difficult to implement advanced mobile object tracking that requires significant computing power due to energy constraints. To ease the energy consumption burden of IoT devices, they proposed an edge computing-based multivariate time series (ET-MTS) framework in IoT systems for accurately tracking mobile objects using edge computing. The EC-MTS framework leverages vector automatic regression to revisit arbitrary historical object trajectory data and fit the best trajectory model for accurate mobile object position prediction. Simulation results demonstrated that EC-MTS showed better object trajectory goodness-of-fit and object location prediction accuracy.

Tarahomi et al. [29] have presented an efficient power-aware virtual machine (VM) allocation algorithm in a cloud data center. They considered virtualization to allocate power-aware VMs because the power efficiency of cloud servers in a data center is very important. The evolutionary algorithms were used for VM allocation to choose a suitable VM as a physical host. Simulation results demonstrated that their proposed algorithms improved power consumption.

Amanatidis et al. [30] have designed a cooperative task execution mechanism for object detection using an edge computing structure. Cooperation with edge computing is essential to realize object detection algorithms in IoT devices with high performance and low power consumption. In their proposed cooperative task execution mechanism, the edge device collaboratively performs object detection on different images from end devices for optimizing the execution time and the execution accuracy. In particular, they focused on new policy that considers image compressing and the batch sizes of the received images from the end devices. Simulation results demonstrated that the proposed policy based on different splitting decisions and compression values can improve the E2E execution time and accuracy.

Most recent studies [27–29] have focused mainly on offloading specific functions on edge and cloud servers rather than selective offloading [21,30] according to system environments to improve inference accuracy and latency performance. In addition, offloading policies are derived based on the actual observed inference latency [30] or the inference latency calculated in proportion to the relevant data size [21]. Since these types of inference latency are somewhat different from the actual inference latency, there is a limit to deriving optimal offload policies for edge computing. We focus on this problem and address, in this paper, inference latency prediction considering the characteristics of inference models, images, and devices. By using the predicted inference latency close to the actual inference latency, the performance improvement by offloading can be maximized. To the best of our knowledge, the inference latency prediction approach is the first methodology introduced in object detection using edge computing.

3. System Model

We consider an AI service model for object detection in edge computing as shown in Figure 1. For the edge computing environment, the hierarchy consisting of the client, edge, and cloud is adopted as a system model. The client device requests an object detection service and can execute this task locally or remotely through network-connected devices by considering various situations as in [11–21]. For remote execution, we assume that the edge device is set to have higher computing power than the client device, and the cloud server is set to have higher computing power than the edge device. To select a device in the client to execute object detection tasks from among the client, edge, and cloud, not only the communication latency but also the inference latency should be considered. Among these, we present an approach to predict the inference latency of the edge and cloud on the client.

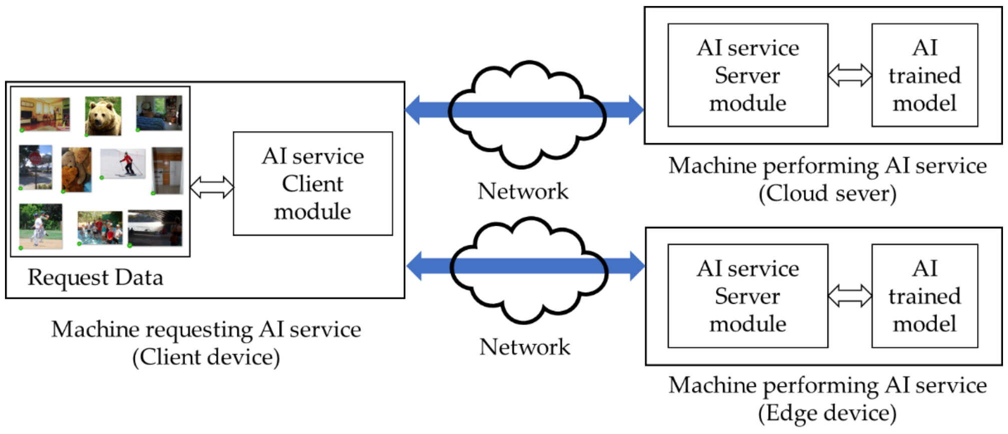


Figure 1. System model for object detection in edge computing.

3.1. System Configuration

To configure the edge computing environment, we considered three machines, corresponding to the client, edge, and cloud, as shown in Table 1. Machine 1, with the best computing power, was used as the cloud server, and machine 3, with the lowest computing power, was used as the client device. Also, machine 2, with moderate computing power, was used as the edge device. Ubuntu version 20.04 (Ubuntu, London, UK) was used along with the Python implementation of the TensorFlow object detection API [31]. When an object detection service was requested from an AI service client module to an AI service server module, REST and gRPC were used as the communication protocols.

Table 1. Specification of machines.

Machine	CPU	GPU	Geekbench 5 Score (Multi-Core)
Machine 1 (Cloud)	Intel Core i7-11700K	RTX 3060	10822
Machine 2 (Edge)	Intel Core i7-11700K	-	10821
Machine 3 (Client)	Intel Core i7-10700K	-	8948

3.2. Object Detection Models

Two types of object detectors are considered in this paper. The SSD-MobileNet [23] and RFCN [22] are considered for the single-stage and two-stage deep object detectors, respectively. The SSD-MobileNet conducts object detection as a simple regression problem by learning the class probabilities and boundary box coordinates of input images. The RFCN generates regions of interest in the first stage, and object classification and boundary box regression are performed on the regions of interest in the second stage. Therefore, the RFCN is more accurate and requires a longer computational time, while the SSD-MobileNet is much faster but provides less accurate results.

3.3. Dataset

For inference latency analysis, the COCO 2017 validation dataset [32], with 5000 images, was used to measure the inference latency of the RFCN and SSD-MobileNet algorithms. We selected ten image samples out of 5000 images as shown in Figure 2 and showed the results of object detection through the RFCN and SSD-MobileNet in Figure 3. The object detection service using the RFCN model shows that most objects on the image are detected with high accuracy (95% or more), but the SSD-MobileNet model shows that only major objects on the image are detected, small objects are not detected, and the detection accuracy of major objects is lower than that in the RFCN model.

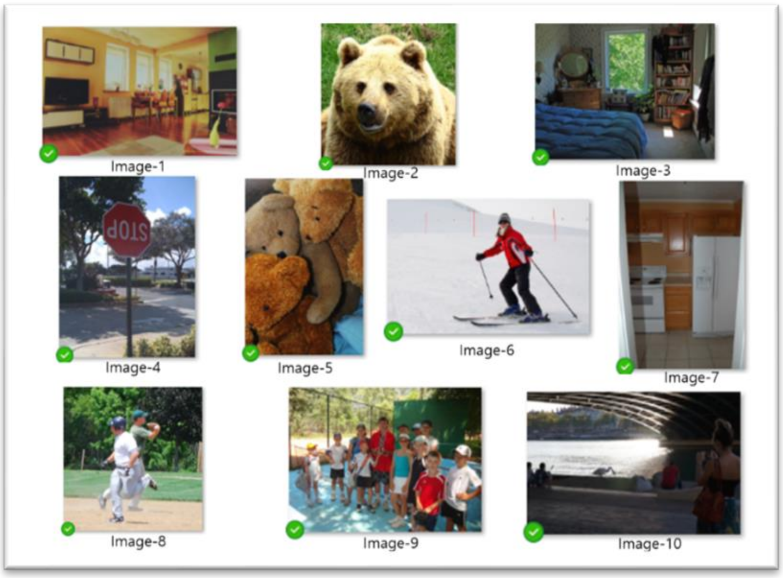
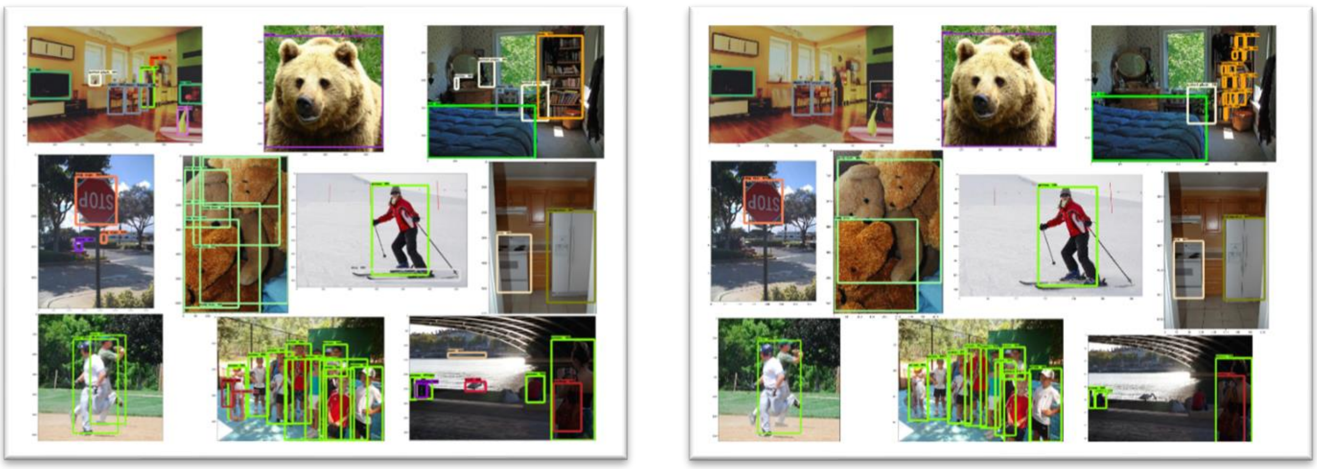


Figure 2. 10 Image samples from the COCO 2017 validation dataset.



(**a**) RFCN (**b**) SSD-MobileNet

Figure 3. Results of object detection for ten image samples: (**a**) RFCN; (**b**) SSD-MobileNet.

4. Proposed Inference Latency Prediction Approaches

This section describes approaches to predict inference latency during object detection in edge computing. In an edge computing environment consisting of a client, edge, and cloud, the two proposed methods approximately predict the inference latency of a task between devices and object detection algorithms by using statistical information. Through experiments, in object detection, we found that the inference latency is not proportional to the size of the image and is determined by the characteristics of the image. In addition, it was confirmed that the tendency of inference latency varies depending on the object detection algorithm in use. Therefore, it is difficult to predict the inference latency of the edge and cloud using only the size of the image. Through various analyses, it was observed that there was a statistical similarity between the inference latencies of devices and object detection algorithms. Using this characteristic, the client can predict the inference latency for the edge and cloud based on the statistical information on the inference latency in the edge and cloud. The predicted inference latency could be used in a scheduling step of determining whether to perform the task locally or remotely on the edge and cloud. After that, the inference latency of object detection was measured for 50 image samples out of 5000 images selected for visualization, and the results are presented in the figures below.

4.1. Inference Latency According to Image Size

This section presents the relationship between image size and inference latency when using the RFCN and SSD-MobileNet models for object detection. In [21], an approach where the inference latency is also increased in proportion to the increase in the image size was considered. However, this approach does not apply to all applications, and an approach suitable for specific application characteristics should be considered.

We measured the inference latency for images with various sizes to analyze the relationship between image size and inference time for the RFCN and SSD-MobileNet models. As shown in Figure 4, we can confirm that the relationship between image size and inference latency is uncorrelated, which shows that it is difficult to predict inference latency with image size alone in object detection.

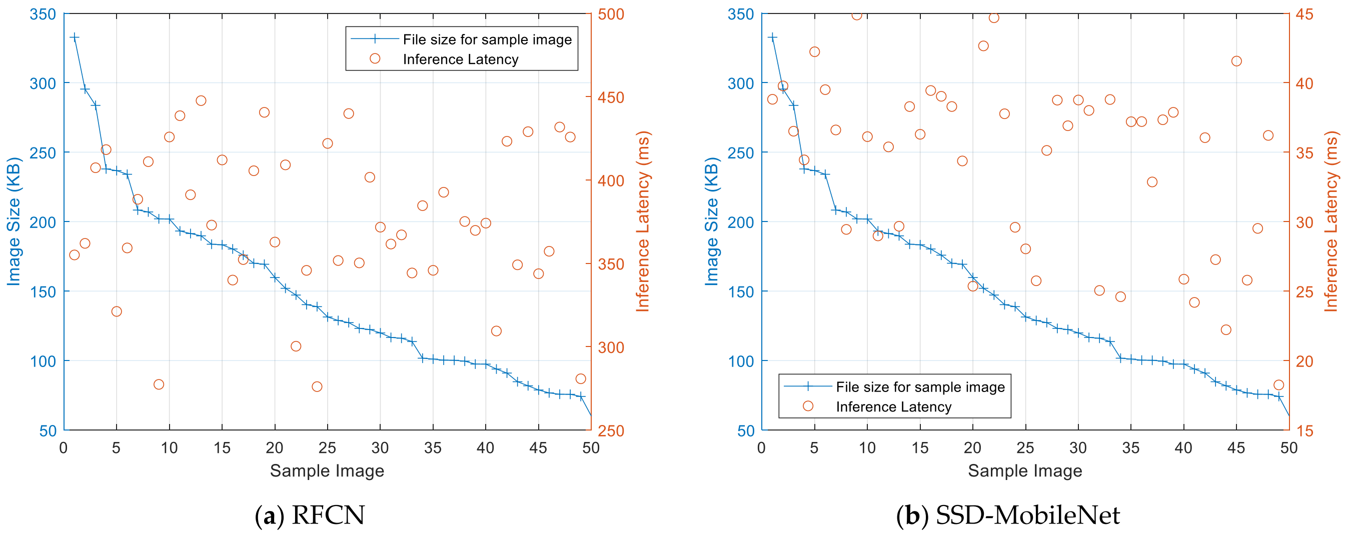
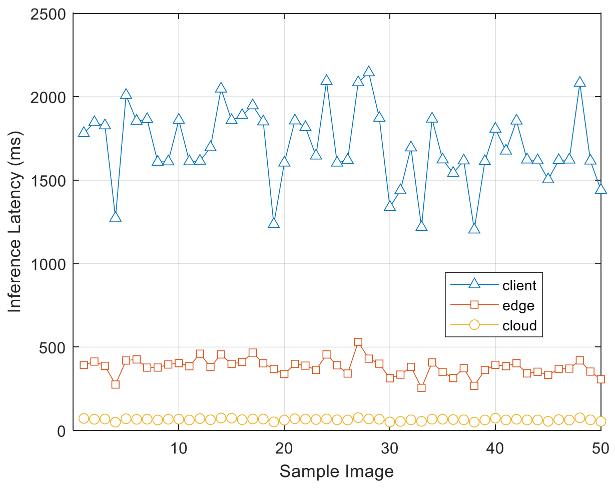


Figure 4. Relationship between inference latency and image size with REST at the edge: (**a**) RFCN; (**b**) SSD-MobileNet.

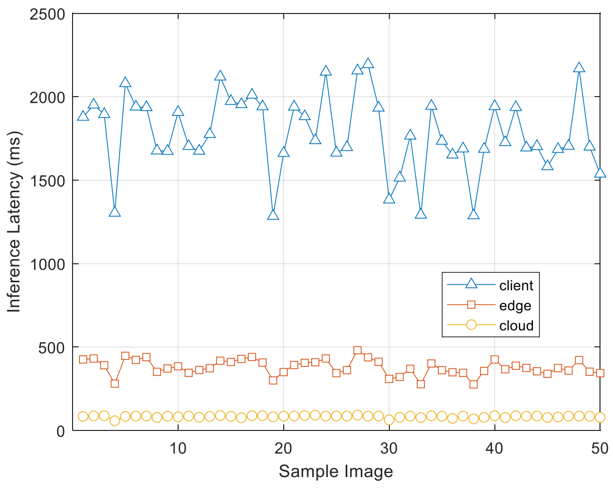
For this reason, the client cannot predict the inference latency required for object detection in the edge and cloud devices, leading to a problem in which it cannot determine where to execute the task, locally or remotely, on the edge and cloud. In other words, when exploiting object detection services in edge computing, the process of predicting inference latency at the edges and clouds should be additionally considered.

4.2. Inference Latency Depending on Devices

This section deals with the relationship between computing power and inference latency when using the RFCN and SSD-MobileNet models for object detection. We measured the inference latency for the gRPC and REST communication protocols depending on three machines with different computing powers as described in Table 1. As shown in Figures 5 and 6, we can observe that the SSD-MobileNet model has a much shorter inference latency than the RFCN model and that gRPC is slightly faster than REST. It can also be seen that the cloud shows the lowest inference latency, the edge shows a slightly longer inference latency than the cloud, and the client shows the longest inference latency. What we should pay attention to here is the tendency of inter-machine inference latency. For example, as shown in Figure 6b, although there are large differences between the absolute inference latencies of the client, edge, and cloud devices, it can be observed that the inference latencies of the client, edge, and cloud devices are relatively similar. In addition, the similarity is more evident in REST than in gPRC. Leveraging this similarity may help the client predict the inference latency of the edge and of the cloud.

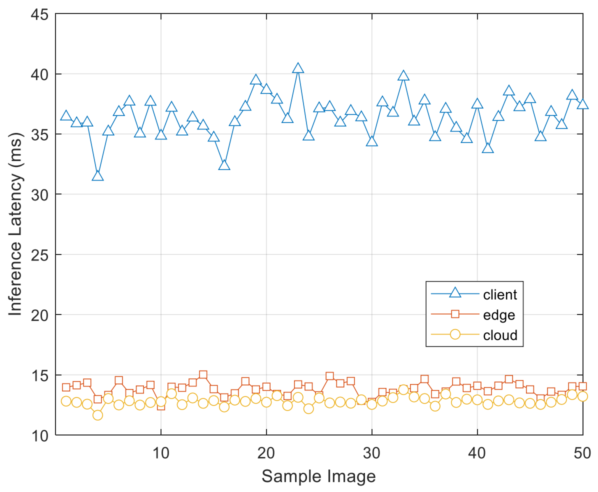


(**a**) gRPC

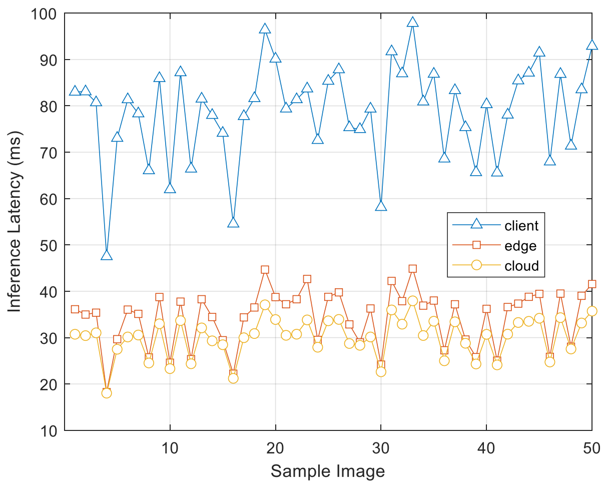


(**b**) REST

Figure 5. Inference latency with the RFCN model for the client, edge, and cloud devices: (**a**) gRPC; (**b**) REST.



(**a**) gRPC



(**b**) REST

Figure 6. Inference latency with the SSD-MobileNet model for the client, edge, and cloud devices: (**a**) gRPC; (**b**) REST.

4.3. Proposed Inference Latency Prediction between Devices

This section introduces the first method (Method 1), wherein the client predicts the inference latency of the edge and cloud with a given inference model. To analyze the similarity in inference latency between devices, we compare the inference latencies of devices by using the normalized inference latency, which converts the distribution of inference latency to a normal distribution. Let X be the distribution of the inference latency for each device. Then, the distribution of normalized inference latency Z can be converted as below:

$$Z = (X - \mu)/\sigma, \tag{1}$$

where μ and σ are the mean and standard deviation of the inference latency for each device for 5000 images. The notations used in this paper are given in Table 2. From now on, the device information is used as a subscript under the symbols of the mean and standard deviation. For example, μ_{cloud} and σ_{cloud} are the mean and standard deviation of the cloud device, respectively. The statistical information is listed in Tables 3 and 4 for the RFCN and SSD-MobileNet models, respectively.

Table 2. List of notations.

Notations	Meaning
X	Distribution of the inference latency for each device
Z	Distribution of the inference latency for each device
μ	Mean of the inference latency for each device
σ	Standard deviation of the inference latency for each device
ρ	Correlation coefficient
α	Factor for reflecting the correlation coefficient
$\hat{X}_{client}$	Distribution of predicted inference latency for client
$\hat{X}_{edge}$	Distribution of predicted inference latency for edge
$\hat{X}_{cloud}$	Distribution of predicted inference latency for cloud
$Z^{SSD-MobileNet}_{client}$	Distribution of normalized latency of SSD-MobileNet for client
Z^{RFCN}_{client}	Distribution of normalized latency of the RFCN for client
Z^{RFCN}_{edge}	Distribution of normalized latency of the RFCN for edge
Z^{RFCN}_{cloud}	Distribution of normalized latency of the RFCN for cloud
$\hat{X}^{RFCN}_{client}$	Distribution of predicted latency of the RFCN for client
$\hat{X}^{RFCN}_{edge}$	Distribution of predicted latency of the RFCN for edge
μ^{RFCN}_{client}	Distribution of predicted latency of the RFCN for cloud

Table 3. Mean and standard deviation values of inference latency for the RFCN model.

(μ, σ)	**Client**	**Edge**	**Cloud**
gRPC	(1705.08, 229.42)	(380.57, 51.08)	(65.11, 6.60)
REST	(1780.61, 231.88)	(378.69, 46.60)	(83.55, 6.64)

Table 4. Mean and standard deviation values of inference latency for the SSD-MobileNet model.

(μ, σ)	**Client**	**Edge**	**Cloud**
gRPC	(36.40, 1.71)	(13.83, 0.56)	(12.81, 0.35)
REST	(78.68, 10.47)	(34.24, 6.24)	(30.09, 4.28)

We compare the similarities of the normalized inference latency values by reflecting the μ and σ of inference latency for the client, edge, and cloud devices. Figures 7 and 8 show the normalized inference latency with the RFCN and SSD-MobileNet models for the client, edge, and cloud devices, respectively. As shown in Figures 7 and 8, we can observe that the normalized inference latency values are similar for the client, edge, and cloud devices. Also, the similarity between the client, edge, and cloud devices is more clearly revealed in REST than in gRPC. Therefore, it can be considered that the inference latency of the client can be utilized to predict the inference latency of the edge and cloud.

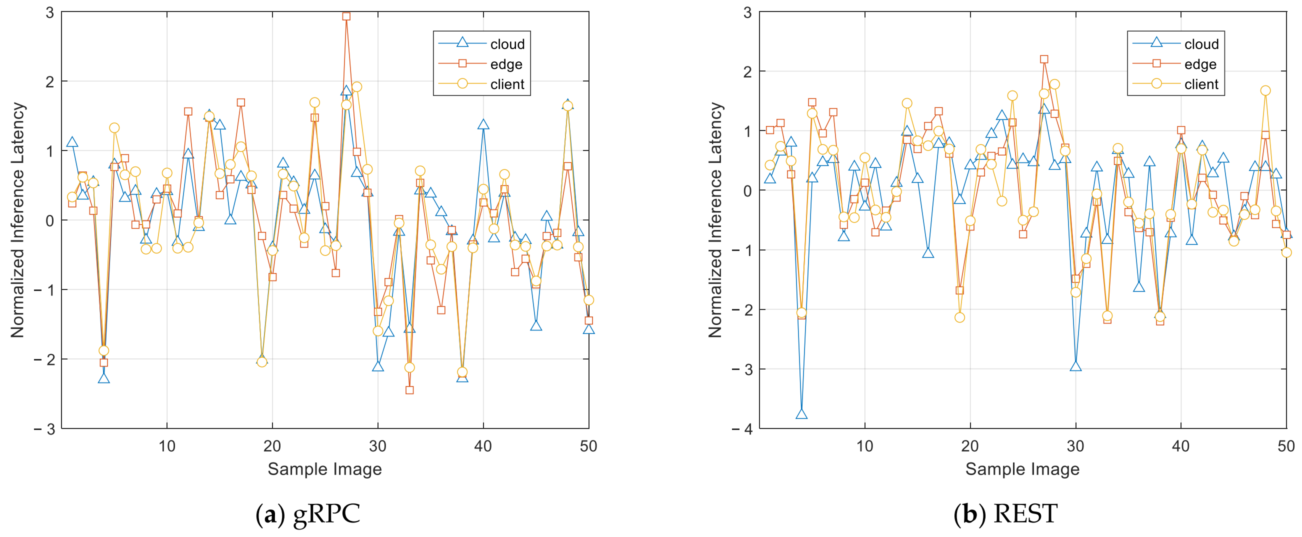


Figure 7. Normalized inference latency with the RFCN model for the client, edge, and cloud devices: (**a**) gRPC; (**b**) REST.

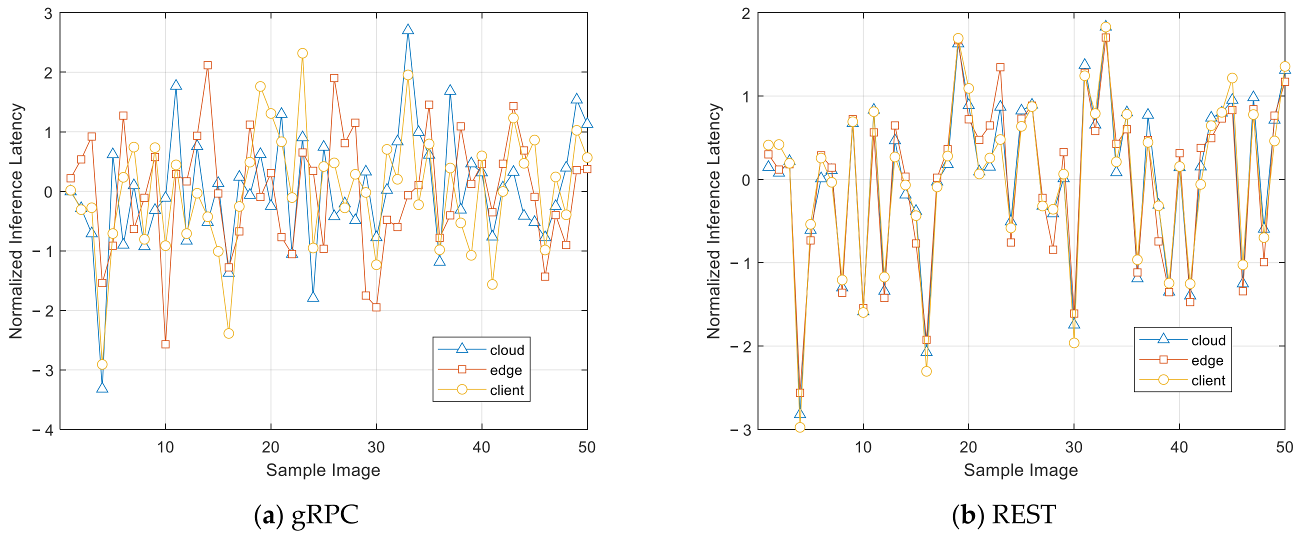


Figure 8. Normalized inference latency with the SSD-MobileNet model for the client, edge, and cloud devices: (**a**) gRPC; (**b**) REST.

Considering the similarity in normalized inference latency between the client, edge, and cloud devices, we propose a method for predicting the inference latency at the edges and clouds for the clients. The predicted inference latency for the edge and cloud in object detection can be obtained by using the normalized inference latency for the client and statistical information for the edge and cloud, as shown in Figure 9.

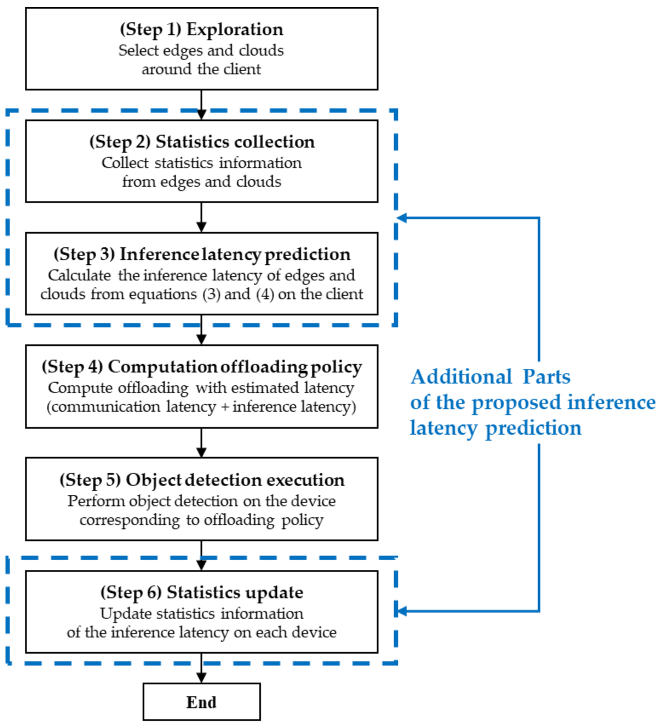

(Step 1) Exploration
Select edges and clouds around the client
(Step 2) Statistics collection
Collect statistics information from edges and clouds
(Step 3) Inference latency prediction
Calculate the inference latency of edges and clouds from equations (3) and (4) on the client
(Step 4) Computation offloading policy
Compute offloading with estimated latency (communication latency + inference latency)
(Step 5) Object detection execution
Perform object detection on the device corresponding to offloading policy
(Step 6) Statistics update
Update statistics information of the inference latency on each device
End
Additional Parts of the proposed inference latency prediction

Figure 9. The flowchart of the proposed inference latency prediction.

Let Z_{client} be the distribution of the normalized inference latency for the client device. This distribution can be easily obtained by executing object detection in the client. Then, the distributions of predicted inference latency for the edge and cloud $\hat{X}_{edge}$ and $\hat{X}_{cloud}$ can be calculated as below:

$$\hat{X}_{edge} = \sigma_{edge} \cdot Z_{client} + \mu_{edge}, \tag{2}$$

$$\hat{X}_{cloud} = \sigma_{cloud} \cdot Z_{client} + \mu_{cloud}. \tag{3}$$

We assumed that the client was informed of the μ and σ of the edge and cloud in advance. To employ the proposed technique, a process of calculating the μ and σ for the edge and cloud according to the application and dataset is required.

From Equations (2) and (3), the client can predict the inference latency of the edge and cloud. The predicted inference latencies for the edge and cloud are utilized as important parameters in determining whether the client executes the object detection task locally or remotely. In applications such as autonomous driving, which is very sensitive to latency, the communication latency and inference latency should be considered more importantly. Therefore, the proposed inference latency prediction technique is expected to play an important role in various applications.

Additionally, the inference latency of the client and cloud is predictable at the edge. Let Z_{edge} be the distribution of the normalized inference latency for the edge device. Then, the distributions of predicted inference latency for the client and cloud $\hat{X}_{client}$ and $\hat{X}_{cloud}$ can be computed as below:

$$\hat{X}_{client} = \sigma_{client} \cdot Z_{edge} + \mu_{client}, \tag{4}$$

$$\hat{X}_{cloud} = \sigma_{cloud} \cdot Z_{edge} + \mu_{cloud}. \tag{5}$$

When it is difficult for the client to determine the device to execute the task, it is possible to predict the inference latency of other devices at the edge and cloud for scheduling.

The steps of the proposed methods are as follows. First, the client searches for edge and cloud devices that are around (Step 1) and receives their statistical information on inference latency (Step 2). Then, the inference latency is predicted based on Z_{client} and the received statistical information on the edge and cloud devices from Equations (2) and (3) on the client (Step 3). Considering the predicted inference latency and communication latency of the edge and cloud devices, the client derives an offloading policy regarding to which device to offload, as in [21] (Step 4). Finally, the workload is offloaded to the selected devices, and they execute object detection (Step 5) and update statistics based on the actual inference latency (Step 6).

4.4. Proposed Inference Latency Prediction between Object Detection Algorithms

This section describes the second method (Method 2), wherein the client with the SSD-MobileNet model predicts the inference latency of the RFCN model for the client, edge, and cloud. It is very efficient to predict the inference latency of the RFCN model through the SSD-MobileNet model because the SSD-MobileNet model operates much faster than the RFCN model, as explained in Section 3.2. To analyze the relationship between the inference latencies of the RFCN and SSD-MobileNet models, we compare the correlation coefficients ρ between the normalized inference latency of the SSD-MobileNet model for the client, $Z_{client}^{SSD-MobileNet}$, and the normalized inference latency of the RFCN model for the client, edge, and cloud, i.e., Z_{client}^{RFCN}, Z_{edge}^{RFCN}, and Z_{cloud}^{RFCN}, as listed in Table 5. The two protocols show somewhat different characteristics. In gRPC, $Z_{client}^{SSD-MobileNet}$ is observed to have a negative correlation with Z_{client}^{RFCN}, Z_{edge}^{RFCN}, and Z_{cloud}^{RFCN}, and in REST, $Z_{client}^{SSD-MobileNet}$ is observed to have a negative correlation with Z_{client}^{RFCN} and Z_{edge}^{RFCN} and a positive correlation with Z_{cloud}^{RFCN}. Therefore, it is possible to predict the inference latency of the RFCN model through that of the SSD-MobileNet model by exploiting these correlation characteristics.

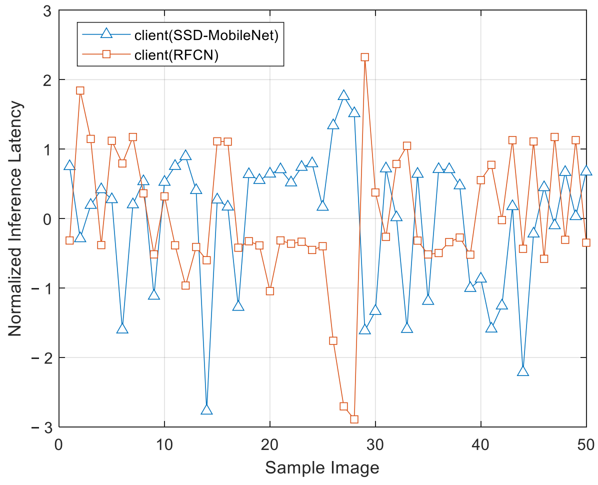
Table 5. The correlation coefficients of normalized inference latency between the SSD-MobileNet model for the client and the RFCN model for the client, edge, and cloud.

ρ	**Client**	**Edge**	**Cloud**
gRPC	–0.39	–0.36	–0.26
REST	–0.39	–0.32	0.38

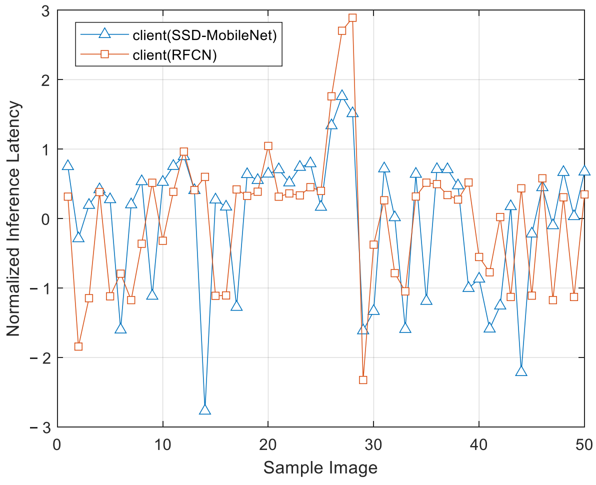
We compared the relationship between the normalized inference latencies of the SSD-MobileNet and RFCN models by considering a factor, α, to reflect the negative correlation. For the positive and negative correlations, α was set to +1 and -1, respectively. Then, the normalized inference latencies of the RFCN model for the client, edge, and cloud, Z_{client}^{RFCN}, Z_{edge}^{RFCN}, and Z_{cloud}^{RFCN}, are predicted as $\alpha \cdot Z_{client}^{SSD-MobileNet}$. Therefore, with the gRPC protocol, the normalized inference latencies of the RFCN model for the client, edge, and cloud are predicted as $-Z_{client}^{SSD-MobileNet}$, $-Z_{client}^{SSD-MobileNet}$, and $Z_{client}^{SSD-MobileNet}$, respectively.

Figure 10 shows the normalized inference latencies of the RFCN and SSD-MobileNet models for the client in the REST protocol. It can be observed that the normalized inference latencies of the client in the RFCN and SSD-MobileNet models have a negative correlation as shown in Figure 10a. By setting α as –1, we can observe that the normalized inference latencies of the client tend to be similar for the RFCN and SSD-MobileNet models, as shown in Figure 10b.

Figure 11 shows the normalized inference latencies of the RFCN model for the edge device and the SSD-MobileNet model for the client device in the REST protocol. Similar to what is observed in the relationship between the RFCN and SSD-MobileNet models for the client device, it can be observed that the normalized inference latencies of the RFCN model for the edge device and the SSD-MobileNet model for the client device have a negative correlation, as shown in Figure 11a. Therefore, it is possible to approximately predict the normalized inference latency of the RFCN model for the edge device by multiplying the normalized inference latency of the SSD-MobileNet model for the client by -1, as shown in Figure 11b. Similarly, the normalized inference latency of the RFCN for the cloud device can be predicted based on the normalized inference latency of the SSD-MobileNet model for the client device.

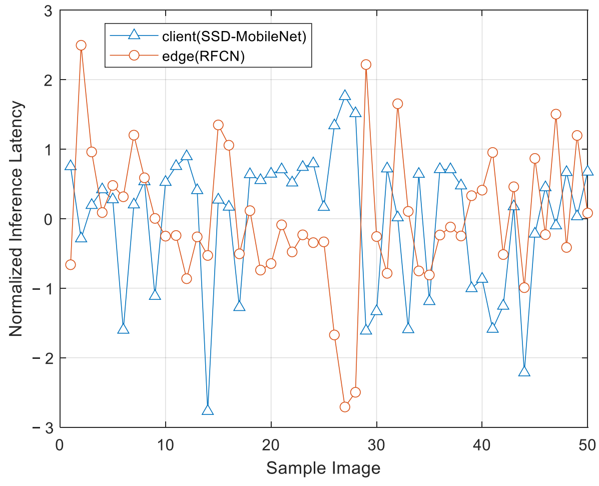


(**a**) $\alpha = 1$

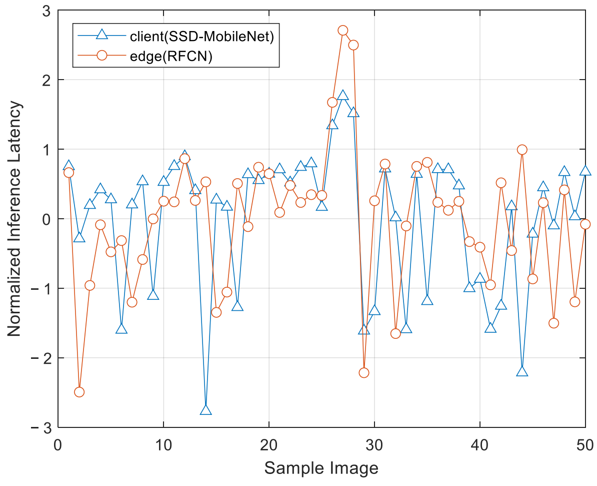


(**b**) $\alpha = -1$

Figure 10. The normalized inference latencies of the RFCN and SSD-MobileNet models for the client in the REST protocol: (**a**) $\alpha = 1$; (**b**) $\alpha = -1$.



(**a**) $\alpha = 1$



(**b**) $\alpha = -1$

Figure 11. The normalized inference latencies of the RFCN model for the edge and the SSD-MobileNet model for the client in the REST protocol: (**a**) $\alpha = 1$; (**b**) $\alpha = -1$.

Considering these similar characteristics of the normalized inference latency between devices, the actual inference latency of the RFCN model for the client, edge, and cloud can be predicted based on their statistical information, α, and $Z_{client}^{SSD-MobileNet}$. First, the SSD-MobileNet model is executed on the client to obtain $Z_{client}^{SSD-MobileNet}$, and α is determined by considering the correlation coefficients of normalized inference latency between the RFCN and SSD-MobileNet models. Then, the distributions of predicted inference latency for the client, edge, and cloud $\hat{X}_{client}^{RFCN}$, $\hat{X}_{edge}^{RFCN}$ and $\hat{X}_{cloud}^{RFCN}$ can be calculated as below:

$$\hat{X}_{client}^{RFCN} = \sigma_{client}^{RFCN} \cdot \left(\alpha \cdot Z_{client}^{SSD-MobileNet}\right) + \mu_{client}^{RFCN}, \tag{6}$$

$$\hat{X}_{edge}^{RFCN} = \sigma_{edge}^{RFCN} \cdot \left(\alpha \cdot Z_{client}^{SSD-MobileNet}\right) + \mu_{edge}^{RFCN}, \tag{7}$$

$$\hat{X}_{cloud}^{RFCN} = \sigma_{cloud}^{RFCN} \cdot \left(\alpha \cdot Z_{client}^{SSD-MobileNet} \right) + \mu_{cloud}^{RFCN}, \tag{8}$$

where μ_{device}^{RFCN} and σ_{device}^{RFCN} are the mean and standard deviation of the inference latency of RFCN model for each device. It is assumed that the μ_{device}^{RFCN} and σ_{device}^{RFCN} for all devices are known in advance on the client and are updated periodically. The correlation characteristics of the inference latency for each inference model and the analysis and delivery of statistical information are not the scope of this paper, and we will further study these topics in the future. As it becomes possible to predict the inference latency of the RFCN model for the client, edge, and cloud devices through the normalized inference latency of the SSD-MobileNet model for the client device, the actual inference latency of the RFCN model with relatively long latency can be predicted as short latency.

5. Simulation Results

This section shows the simulation results of the two proposed inference latency prediction methods in object detection. While the proposed method 1 aims to predict the inference latency between devices using a given inference model, the proposed method 2 aims to predict the inference latency between a complex two-stage and a simple single-stage inference model. For object detection, the RFCN and SSD-MobileNet are considered as two-stage and single-stage object detectors, respectively, and gRPC and REST are used as the communication protocols. We evaluate that the proposed inference latency prediction methods work well in various environments through experimental results. For prediction accuracy comparison, the root mean square error (RMSE) is calculated between the distribution of inference latency X and the distribution of predicted inference latency $\hat{X}$.

5.1. Performance Evaluation of Proposed Method 1 with RFCN Model

We measured the predicted inference latency of the proposed method 1 in the RFCN model and now compare it to the actual inference latency for the edge and cloud. Figures 12 and 13 show the actual and predicted inference latency with the RFCN model for the cloud and edge, respectively. Overall, as shown in Figures 12 and 13, it can be observed that the predicted inference latency with the RFCN model reflects the actual inference latency for the edge and cloud well. Therefore, it can be confirmed that the proposed inference latency prediction approach is suitable for object detection.

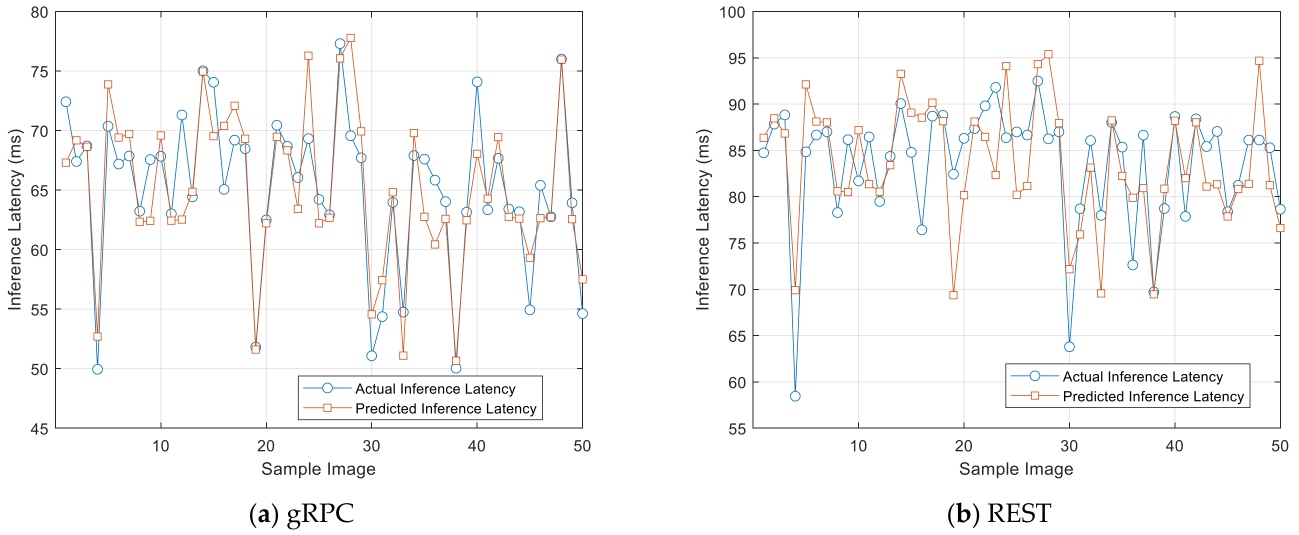
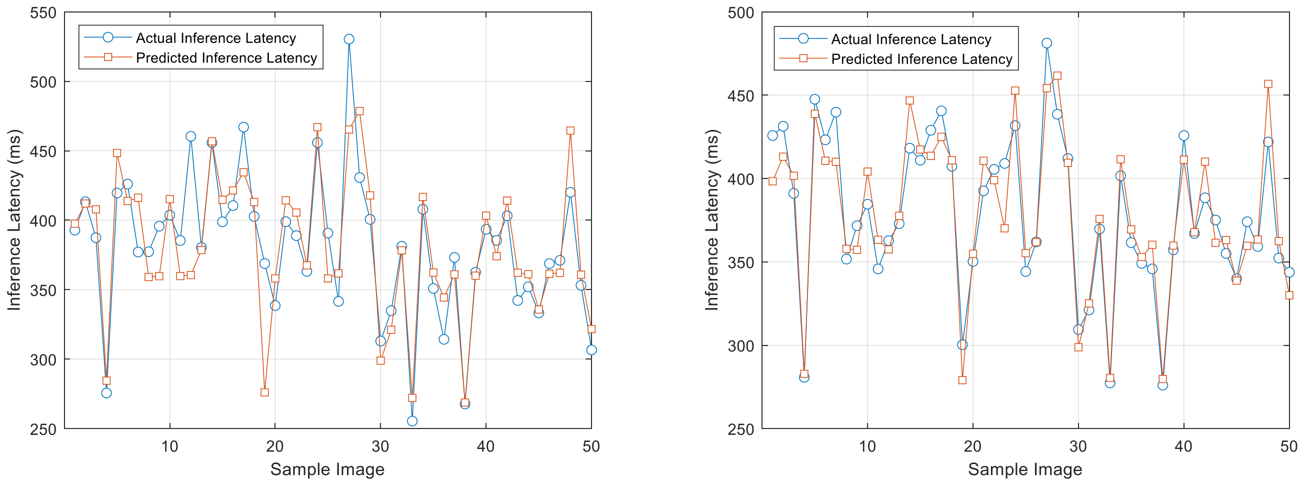


Figure 12. The actual and predicted inference latency of method 1 with the RFCN model for the cloud: (**a**) gRPC; (**b**) REST.



(**a**) gRPC (**b**) REST

Figure 13. The actual and predicted inference latency of method 1 with the RFCN model for the edge: (**a**) gRPC; (**b**) REST.

When comparing the results for the edge and cloud, the inference latency of the cloud is relatively very low, and so the similarity between the predicted and actual inference latency at the edge is longer than that of the cloud. Although the difference between the predicted and actual inference latency seems small at the edge, the RMSE of the edge is larger than that of the cloud, as shown in Table 6, since the inference latency at the edge is relatively long. The difference between the predicted and actual inference latency for the cloud seems more noticeable than for the edge. However, the RMSE of the cloud is smaller than that of the edge.

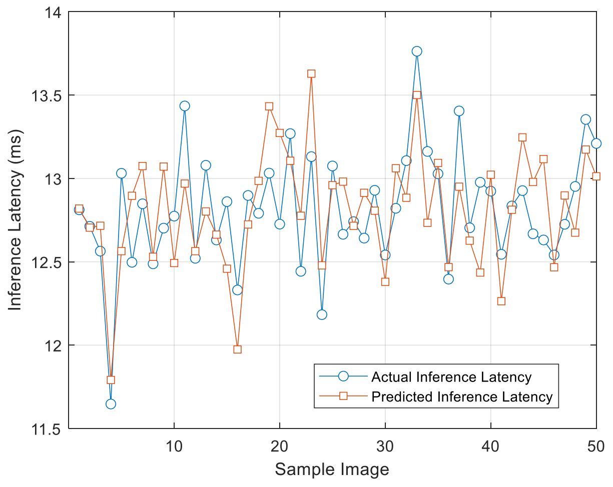
Table 6. The RMSE performance of method 1 with the RFCN model for the edge and cloud.

RMSE	Cloud	Edge
gRPC	3.259	28.300
REST	5.372	15.554

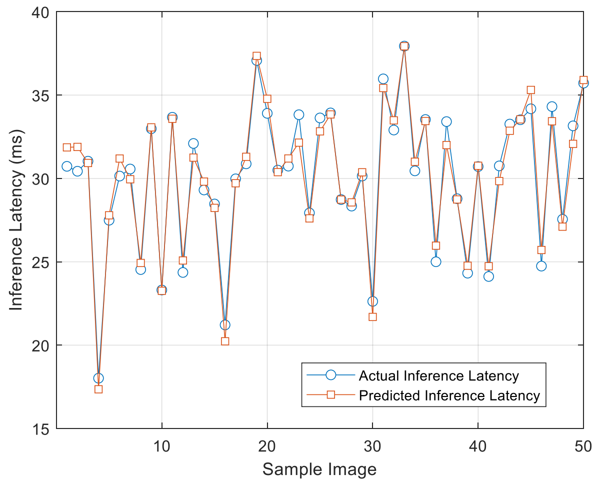
A comparison was also conducted according to the communication protocols gRPC and REST. Two different trends have been observed depending on the communication protocol. First, the RMSE performance of gRPC is lower than that of REST in the cloud. In other words, the predicted inference latency is more accurate when gRPC is used than when REST is used in the cloud. Second, the RMSE performance of REST is lower than that of gRPC at the edge. That is, the predicted inference latency is more accurate when REST is used than when gRPC is used at the edge. Figures 12 and 13 also confirm that using gRPC in the cloud and REST at the edge is better for predicting inference latency.

5.2. *Performance Evaluation of Proposed Method 1 with SSD-MobileNet Model*

We measured the predicted inference latency of the proposed method 1 in the SSD-MobileNet model and now compare it to the actual inference latency for the edge and cloud. Figures 14 and 15 show the actual and predicted inference latency with the SSD-MobileNet model for the cloud and edge, respectively. Overall, as shown in Figures 14 and 15, it can be observed that the predicted inference latency with the SSD-MobileNet model reflects the actual inference latency for the edge and cloud well. Therefore, it can be also confirmed that the proposed inference latency prediction approach is suitable for object detection.

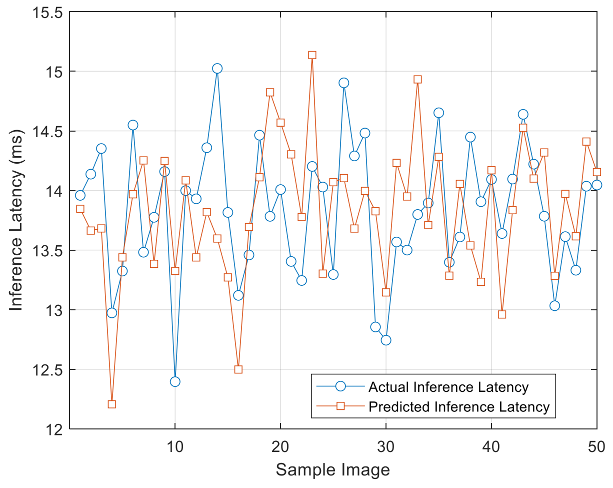


(**a**) gRPC

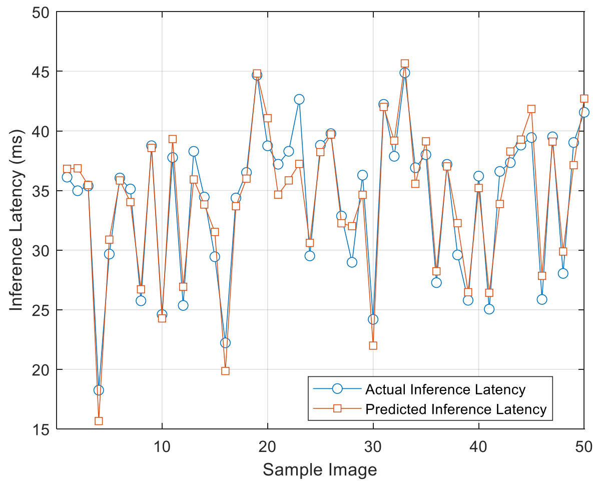


(**b**) REST

Figure 14. The actual and predicted inference latency of method 1 with the SSD-MobileNet model for the cloud: (**a**) gRPC; (**b**) REST.



(**a**) gRPC



(**b**) REST

Figure 15. The actual and predicted inference latency of method 1 with the SSD-MobileNet model for the edge: (**a**) gRPC; (**b**) REST.

The SSD-MobileNet model has a tendency that is different from the results of the RFCN model because the difference in inference latency between the cloud and edge is small. In the SSD-MobileNet model, there is a large difference in terms of inference latency depending on the communication protocol rather than the difference in inference latency according to the device. When comparing the results of gRPC and REST, the similarity between the predicted and actual inference latency while using REST is greater than that while using gRPC. However, the RMSE of gRPC is much smaller than that of REST, as shown in Table 7.

Table 7. The RMSE performance of method 1 with the SSD-MobileNet model for the edge and cloud.

RMSE	Cloud	Edge
gRPC	0.285	0.606
REST	0.689	1.705

There is a trade-off relationship between gRPC and REST in the SSD-MobileNet model. The gRPC protocol has a lower inference latency and RMSE performance compared to the REST protocol, but it has a lower similarity between the prediction and the actual inference latency. Conversely, the REST protocol has a high inference latency and the RMSE performance compared to the gRPC protocol, but it has a high similarity between the prediction and the actual inference latency. When building a system environment, it is necessary to consider this trade-off relationship.

5.3. Performance Evaluation of Proposed Method 2

We measured the predicted inference latency of the proposed method 2 using the SSD-MobileNet model and now compare it to the actual inference latency of the RFCN model for the client and edge devices. Figure 16 shows the actual and predicted inference latency of the RFCN model for the client and edge devices using the SSD-MobileNet model for the client device in the REST protocol. Although there are more prediction errors when compared to the proposed method 1, it can be observed that the predicted inference latency of the proposed method 2 is also highly similar to the actual inference latency. Simulation results confirm that the use of the characteristics of the negative correlation between the RCFN and SSD-MobileNet models can effectively predict the inference latency of the complex RFCN model. Additionally, the proposed method 2 predicts not only the inference latency of the RFCN model for the client but also the inference latency of the RFCN model for the edge device well.

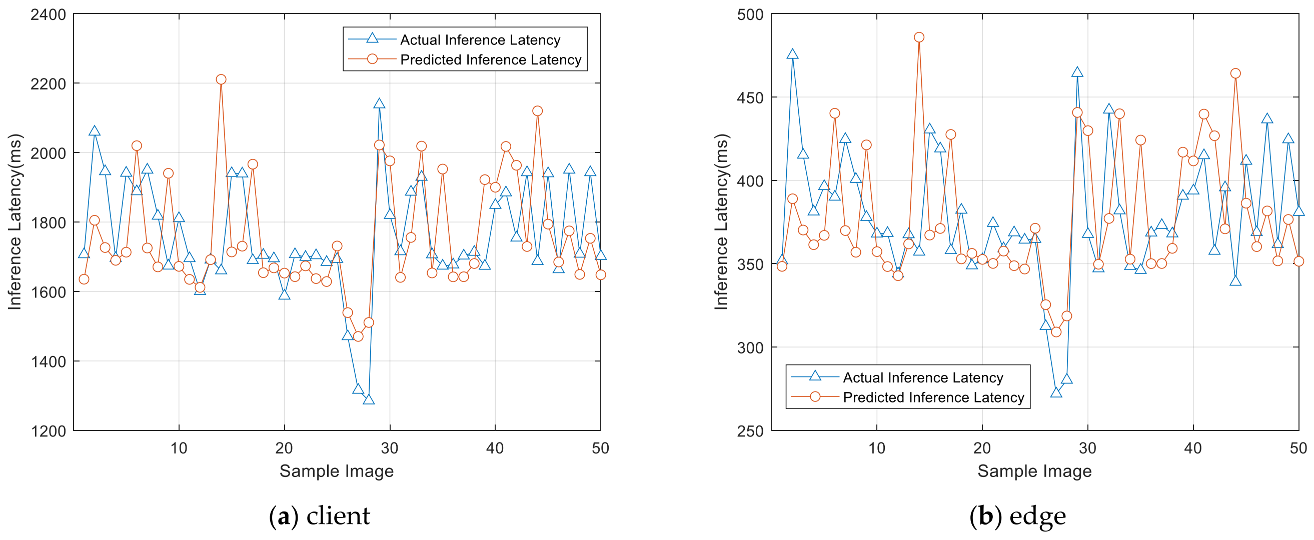


Figure 16. The actual and predicted inference latency of method 2 with the RFCN model for each device using the SSD-MobileNet for the client in the REST protocol: (**a**) client; (**b**) edge.

We also evaluated the RMSE performance of the proposed method 2 in the gRPC and REST protocols as shown in Tables 8 and 9, respectively. When calculating the RMSE, two cases, where α is 1 and -1, in the proposed method 2 were considered to understand the effect of α. According to the correlation given in Table 4, it can be seen that the proposed method 2 with an α value of -1 decreases the RMSE for a negative correlation, and the proposed method 2 with an α value of 1 decreases the RMSE for a positive correlation. By setting α to -1, it was possible to improve the performance by more than 30% from the perspective of the RMSE for negative correlation. Therefore, the proposed inference latency prediction method considering correlation is suitable for predicting the inference latency of the complex RFCN model.

Table 8. The RMSE performance of method 2 in the gRPC protocol.

RMSE	Client	Edge	Cloud
$\alpha = 1$	297.92	63.60	7.70
$\alpha = -1$	196.44	43.23	5.87

Table 9. The RMSE performance of method 2 in the REST protocol.

RMSE	Client	Edge	Cloud
$\alpha = 1$	298.00	60.03	5.05
$\alpha = -1$	196.75	42.74	7.58

5.4. Complexity Analysis of Proposed Inference Latency Prediction Methods

This section describes the complexity of the proposed methods. The proposed methods predict the inference latency through three steps. First, the inference latency is measured on the client device. Then, the normalized inference latency is computed by reflecting the statistical information on the measured inference latency on the client device. Lastly, the predicted inference latencies of the edge and cloud devices are obtained by considering their statistical information on the normalized inference latency. Since the second and third steps involve simple scaling, most of the complexity is due to the first step of measuring the inference latency on the client. However, the first step can also be executed relatively simply because it uses a simple inference model, such as the SSD-MobileNet model, when measuring inference latency on the client device. In addition, since this process can be performed in parallel on a separate core of the client, inference latency can be predicted with very low time overhead if only the statistical information of each device is updated.

6. Conclusions

In this paper, we proposed two inference latency prediction approaches for object detection in edge computing. Predicting accurate inference latency is essential because inference latency plays an important role in deriving the optimal offloading policy for edge computing. Using the proposed method, it was possible to predict the inference latency of other devices in a particular device within a given model by using statistical information on their inference latency. Additionally, the proposed method was also able to predict the inference latency of other devices on a certain device across different inference models. Through various experiments, the performances of actual and predicted inference latency were compared according to the selected object detection algorithms and communication protocols. The simulation results show that the predicted inference latency is well matched with the actual inference latency between devices or algorithms. Therefore, the proposed inference latency prediction approaches are suitable for AI services using object detection in edge computing. In the future, we will study and evaluate the performance of offloading policy decisions by applying the proposed inference latency prediction methods in real system environments.

Author Contributions: Conceptualization, G.K. and Y.-G.H.; methodology, G.K. and Y.-G.H.; software, G.K.; validation, G.K. and Y.-G.H.; formal analysis, G.K.; investigation, G.K. and Y.-G.H.; writing—original draft preparation, G.K.; writing—review and editing, G.K. and Y.-G.H.; supervision, Y.-G.H.; project administration, Y.-G.H.; funding acquisition, Y.-G.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Daejeon University Research Grants (2021).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. In Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, BC, Canada, 6–12 December 2020.
2. Tensorflow Serving. Available online: https://www.tensorflow.org/tfx/guide/serving (accessed on 11 July 2023).
3. TorchServe. Available online: https://pytorch.org/serve/ (accessed on 11 July 2023).
4. Nvidia Trion Server. Available online: https://developer.nvidia.com/nvidia-triton-inference-server (accessed on 11 July 2023).
5. Intel OpenVINO. Available online: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html (accessed on 11 July 2023).
6. Sadatdiynov, K.; Cui, L.; Zhang, L.; Huang, J.Z.; Salloum, S.; Mahmud, M.S. A review of optimization methods for computation offloading in edge computing networks. *Digit. Commun. Netw.* **2022**, *9*, 450–461. [CrossRef]
7. Feng, C.; Han, P.; Zhang, X.; Yang, B.; Liu, Y.; Guo, L. Computation offloading in mobile edge computing networks: A survey. *J. Netw. Comput. Appl.* **2022**, *202*, 103366. [CrossRef]
8. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [CrossRef]
9. Kang, P.; Somtham, A. An evaluation of modern accelerator-based edge devices for object detection applications. *Mathematics* **2022**, *10*, 4299. [CrossRef]
10. Hui, Y.; Lien, J.; Lu, X. Early experience in benchmarking edge AI processors with object detection workloads. *Lect. Notes Comput. Sci.* **2020**, *12093*, 32–48.
11. Liao, Z.; Peng, J.; Xiong, B.; Huang, J. Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm. *J. Cloud Comput.* **2021**, *10*, 1–16. [CrossRef]
12. Xu, Z.; Zhao, L.; Liang, W.; Rana, O.F.; Zhou, P.; Xia, Q.; Xu, W.; Wu, G. Energy-aware inference offloading for DNN-driven applications in mobile edge clouds. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 799–814. [CrossRef]
13. Li, B.; He, M.; Wu, W.; Sangaiah, A.K.; Jeon, G. Computation offloading algorithm for arbitrarily divisible applications in mobile edge computing environments: An OCR case. *Sustainability* **2018**, *10*, 1611. [CrossRef]
14. Dinh, T.Q.; La, Q.D.; Quek, T.Q.; Shin, H. Learning for computation offloading in mobile edge computing. *IEEE Trans. Commun.* **2018**, *66*, 6353–6367. [CrossRef]
15. Zhang, H.; Yang, Y.; Huang, X.; Fang, C.; Zhang, P. Ultra-low latency multi-task offloading in mobile edge computing. *IEEE Access* **2021**, *9*, 32569–32581. [CrossRef]
16. Ale, L.; Zhang, N.; Fang, X.; Chen, X.; Wu, S.; Li, L. Delay-aware and energy-efficient computation offloading in mobile edge computing using deep reinforcement learning. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 881–892. [CrossRef]
17. Yu, S.; Wang, X.; Langar, R. Computation offloading for mobile edge computing: A deep learning approach. In Proceedings of the IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, Canada, 8–13 October 2017; pp. 1–6.
18. Ali, Z.; Jiao, L.; Baker, T.; Abbas, G.; Abbas, Z.H.; Khaf, S. A deep learning approach for energy efficient computational offloading in mobile edge computing. *IEEE Access* **2019**, *7*, 149623–149633. [CrossRef]
19. Shakarami, A.; Shahidinejad, A.; Ghobaei-Arani, M. An autonomous computation offloading strategy in Mobile Edge Computing: A deep learning-based hybrid approach. *J. Netw. Comput. Appl.* **2021**, *178*, 102974. [CrossRef]
20. Irshad, A.; Abbas, Z.H.; Ali, Z.; Abbas, G.; Baker, T.; Al-Jumeily, D. Wireless powered mobile edge computing systems: Simultaneous time allocation and offloading policies. *Electronics* **2021**, *10*, 965. [CrossRef]
21. Abbas, Z.H.; Ali, Z.; Abbas, G.; Jiao, L.; Bilal, M.; Suh, D.-Y.; Piran, M.J. Computational Offloading in Mobile Edge with Comprehensive and Energy Efficient Cost Function: A Deep Learning Approach. *Sensors* **2021**, *21*, 3523. [CrossRef] [PubMed]
22. Dai, J.; Li, Y.; He, K.; Sun, J. R-fcn: Object detection via region-based fully convolutional networks. In Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016.
23. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single shot multibox detector. In Proceedings of the 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016.
24. Vadlamani, S.L.; Emdon, B.; Arts, J.; Baysal, O. Can GraphQL Replace REST? A Study of Their Efficiency and Viability. In Proceedings of the IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP), Madrid, Spain, 4 June 2021; pp. 10–17.
25. REST API (Introduction). Available online: https://www.geeksforgeeks.org/rest-api-introduction/ (accessed on 11 July 2023).
26. gRPC. Available online: https://grpc.io/ (accessed on 11 July 2023).
27. Wu, Y.; Guo, H.; Chakraborty, C.; Khosravi, M.; Berretti, S.; Wan, S. Edge computing driven low-light image dynamic enhancement for object detection. *IEEE Trans. Netw. Sci. Eng.* **2023**. [CrossRef]
28. Wu, Y.; Tian, P.; Cao, Y.; Ge, L.; Yu, W. Edge computing-based mobile object tracking in internet of things. *High.-Confid. Comput.* **2022**, *2*, 100045. [CrossRef]
29. Tarahomi, M.; Izadi, M.; Ghobaei-Arani, M. An efficient power-aware VM allocation mechanism in cloud data centers: A micro genetic-based approach. *Cluster. Comput.* **2020**, *24*, 919–934. [CrossRef]
30. Amanatidis, P.; Karampatzakis, D.; Iosifidis, G.; Lagkas, T.; Nikitas, A. Cooperative task execution for object detection in edge computing: An internet of things application. *Appl. Sci.* **2023**, *13*, 4982. [CrossRef]

31. TensorFlow Object Detection API. Available online: https://github.com/tensorflow/models/tree/master/research/object_detection (accessed on 11 July 2023).
32. COCO Dataset. Available online: https://cocodataset.org/ (accessed on 11 July 2023).