IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 17, 2021, accepted September 30, 2021, date of publication October 8, 2021,
date of current version October 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3118731

Implementing Practical DNN-Based Object Detection Offloading Decision for Maximizing Detection Performance of Mobile Edge Devices

GIHA YOON, GEUN-YONG KIM, HARK YOO, SUNG CHANG KIM, AND RYANGSOO KIM
Honam Research Center (HRC), Electronics and Telecommunications Research Institute (ETRI), Gwangju 61012, Republic of Korea

Corresponding author: Ryangsoo Kim (rskim@etri.re.kr)

This work was supported by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government. [21ZK1100, Honam region regional industry-based ICT convergence technology advancement support project]

ABSTRACT In the last decade, deep neural network (DNN)-based object detection technologies have received significant attention as a promising solution to implement a variety of image understanding and video analysis applications on mobile edge devices. However, the execution of computationally intensive DNN-based object detection workloads in mobile edge devices is insufficient in fulfilling the object detection requirements with high accuracy and low latency, owing to the limited computation capacity. In this paper, we implement and evaluate a DNN-based object detection offloading framework to improve the object detection performance of mobile edge devices by offloading computation-intensive workloads to a remote edge server. However, preliminary experimental results have shown that offloading all object detection workloads of mobile edge devices may lead to worse performance than executing the workloads locally. This degradation is obtained from the inefficient resource utilization in the edge computing architectures, both for the edge server and mobile edge devices. To resolve the aforementioned problem with degradation, we devise a device-aware DNN offloading decision algorithm that is aimed to maximize resource utilization in the edge computing architecture. The proposed algorithm decides whether or not to offload the object detection workloads of edge devices by considering their computing power and network bandwidth, and therefore maximizing their average object detection processing frames per second. Through various experiments conducted in a real-life wireless local area network (WLAN) environment, we verified the effectiveness of the proposed DNN-based object detection offloading framework.

INDEX TERMS Deep learning offloading, object detection, wireless edge computing, resource optimization.

I. INTRODUCTION

With an explosive increase of deep learning technologies in the last decade, object detection with deep neural networks (DNNs) has made a great impact on performance improvement in terms of detection accuracy and response time [1]. Moreover, tremendous efforts have been made to continuously improve the DNN-based object detection capabilities of mobile edge devices to implement various image and video analysis applications, including mobile augmented reality, surveillance drones, and autonomous driving [2]. One way to provide such capabilities is to execute the computationally intensive object detection workloads locally at the mobile edge devices. However, performing DNN inference on mobile edge devices imposes a heavy computational burden, resulting in insufficient object detection capabilities. Therefore, various object detection offloading methods, which offload computationally intensive workloads to remote servers with high computing power, are proposed as promising solutions to overcome this limitation [3]–[5].

The traditional approach for providing powerful computing capabilities to mobile edge devices is to utilize cloud computing services through wireless networks. However, it requires a large volume of data transmission via a long wide-area network, resulting in a long and volatile end-to-end latency [6]. To address this certain drawback, the edge computing paradigm has been employed in object detection offloading strategies [7]. In the edge computing paradigm,

The associate editor coordinating the review of this manuscript and approving it for publication was Pedro R. M. Inácio.

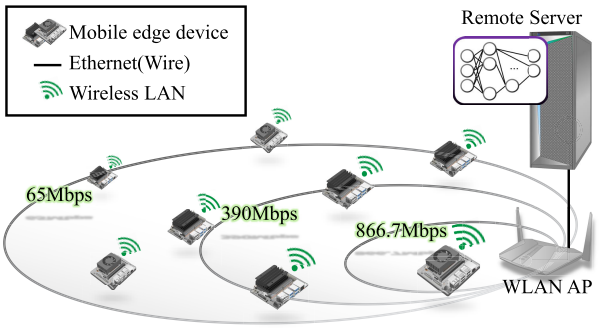


FIGURE 1. Object detection offloading scenario in an edge-computing architecture consisting of multiple mobile edge devices, a remote edge server, and WLAN AP.

the computing capability is located from the network core to the network edge that is in close proximity to the mobile edge devices and is capable of providing low-latency object detection services [8]. As a result, offloading computational tasks to the remote edge server in edge computing architecture is the most promising method to bringing high-level intelligence to the resource-constrained mobile edge devices in various applications that requires DNN-based data analysis such as image classification, action recognition, speech recognition, and anomaly detection [9]–[11].

In this paper, we implement an object detection offloading framework to validate the effectiveness of the object detection offloading service in a real-world wireless local area network (WLAN) environment. The proposed framework is designed to execute computationally intensive workloads, including the DNN inference and non-maximum suppression (NMS) filtering on the edge server, while relatively lightweight workloads, including the resizing images and rendering detection results, are running on mobile edge devices. Furthermore, we apply the task-level pipeline parallelism presented in [12] to improve the computing resource utilization on both the edge devices and edge server, reducing the overall object detection latency.

Despite the enhanced object detection capabilities through our object detection offloading framework, several issues still exist to be considered before applying them to real-world scenarios where multiple mobile edge devices exist as depicted in Fig. 1. The network and computing resource consumption of the edge computing infrastructure worsens as the number of mobile edge devices requesting object detection offload services increases, resulting in an increase in the object detection offload service latency. In addition, time-varying wireless fading environments and unpredictable mobile edge device locations cause irregular fluctuations in data rates, resulting in deviations in the object detection offload service latency among mobile edge devices. In the worst case scenario, which has a relatively small data rate, executing object detection workloads locally may have improved detection performance than offloading them, indicating that offloading object detection workloads of all mobile edge devices does not guarantee the best performance in terms of average frames per second (FPS). These problems are incurred by the inefficient resource utilization in the edge computing architecture, both for the edge server and mobile edge devices. This emphasizes the need for object detection offloading management that is capable of optimizing the resource utilization of the edge computing architecture to maximize the overall detection performance.

We concentrate our attention on the problem of how to decide which mobile edge devices offload their workloads for maximizing average FPS of all the mobile edge devices. In this paper, we focus on a binary object detection offloading method, in which the object detection workloads can be fully offloaded to the edge server or executed locally at the mobile edge devices. Then, we formulate the object detection offloading decision as a binary optimization problem, for which the optimal solution can be obtained by a binary combinatorial algorithm. Note that the binary optimization problem is NP-hard, indicating that it is unable to find a globally optimal solution in polynomial time. To resolve such a drawback, we propose a greedy algorithm that iteratively finds a suboptimal solution in polynomial time. In addition, we mathematically prove a certain condition in which the proposed greedy algorithm always finds the globally optimal solution. We implement the proposed offloading decision algorithm on our object detection offloading framework testbed and dem'onstrate its effectiveness by presenting various experimental results.

The main contributions of this paper are as follows:

- We implement an object detection offloading framework, which is designed to parallelize the workload execution by distributing heavy and lightweight workloads to the edge server and mobile edge devices simultaneously to improve the object detection performance, in a real-world WLAN environment. In addition, we apply object detection-specific task-level pipeline parallelism proposed in our previous work [12] to improve the computing resource utilization on both the mobile edge devices and edge server, reducing the overall object detection latency.
- We derive an object detection offloading decision problem as a simple binary optimization problem and propose a greedy algorithm to find the optimal solution in polynomial time. In addition, we verify the effectiveness of the greedy algorithm by mathematically deriving a certain condition in which the proposed algorithm always finds a globally optimal solution.
- We present the experimental results in various scenarios used in validating the effectiveness of the proposed object detection offloading framework in a real-world WLAN environment, where multiple mobile edge devices are distributed geographically over a WLAN coverage area.

The rest of this paper is organized as follows. Several related works are presented in Section II, while an overview of the proposed object detection offloading framework and its preliminary experimental results are discussed in Section III. The object detection offloading decision problem

is formulated based on the preliminary experimental results, and the object detection offloading decision algorithm is proposed in Section IV. In Section V, the experimental results are presented to demonstrate the effectiveness of the proposed object detection offloading framework in a real-world WLAN environment. Finally, this paper is concluded in Section VI, where the future work is also included.

II. RELATED WORK

In this section, we discuss several previous studies involved in deep learning offloading methods that provide more intelligence to the lightweight and energy-constrained mobile edge devices by exploiting the remote computing resources of edge- and cloud-computing infrastructures.

A. OBJECT DETECTION OFFLOADING FRAMEWORKS

Various edge-assisted object detection offloading frameworks were proposed to improve the deep-learning-based object detection performance in terms of end-to-end latency, detection accuracy, and energy consumption. Ran *et al.* [3] introduced a measurement-driven object detection offloading framework, called the DeepDecision, that decides where and which deep learning model is executed to fulfill the application requirements, including the frame rate, accuracy, and energy consumption. Based on the measured experimental results, they argued that the network transmission latency is the most critical factor that affects frame rates rather than the deep learning processing time executed at the remote server. DeepDecision is capable of determining the optimal offloading decision policy under time-varying network conditions. Liu *et al.* [4] proposed an object detection offloading framework, which decouples the rendering workload from the offloading pipeline to reduce the waiting time for the detection results, for real-time mobile augmented reality (MAR) applications. In addition, another study proposed a split computing framework that aims to reduce the amount of data to be transmitted to the edge server [13]. The framework splits the deep learning model into head and tail models, which are executed at the mobile edge device and remote edge servers in which the output data of the head model are quantized and transmitted to the edge server to ensure that they can be used as input data for the tail model, for a given deep learning model. All frameworks significantly improve the end-to-end latency at the mobile edge device by offloading the object detection workloads to the edge server. However, all these frameworks are designed to support only a single mobile edge device, and thus cannot be easily extended to a more complicated scenario, where multiple mobile edge devices are presented.

Recently, various efforts to design deep learning offloading frameworks have been made to support multiple mobile edge devices. Zhou *et al.* [14] presented a vehicle-to-everything (V2X) framework, called EEVEE, that aims to share augmented contextual information among multiple vehicles on the road by offloading their object detection workloads to the roadside units (RSUs). They devised an edge server selection algorithm operated in a distributed manner to alleviate the latency required for collecting statistics from clients in V2X wireless systems. Meanwhile, Wang *et al.* [5] introduced an energy-aware edge-assisted MAR system that dynamically changes the MAR client configuration parameters, including the CPU frequency and computation model size, to minimize the per-frame energy consumption without latency and accuracy degradation. They proposed a low-energy, accurate, and fast (LEAF) optimization algorithm to determine the optimal MAR configuration parameters and radio resource allocation for multiple MAR clients. Although the frameworks presented in [5] and [14] were designed to find the optimal offloading decision policy for supporting multiple mobile edge devices, they still have several drawbacks that limit their application in a real-world environment, where the network bandwidth and local computing power of the mobile edge devices are different. In this paper, we propose a device-aware deep learning offloading framework that determines the optimal offloading decision policy for multiple mobile edge devices by considering their different network bandwidths, local computing powers, and the degree of congestion at the edge server. To the best of our knowledge, our work is the first to implement a multi-user object detection offloading system and analyze its performance through rigorous experiments applied in the real-world WLAN environment.

B. COMPUTATION OFFLOADING DECISION ALGORITHMS

The increasing interest in edge computing architectures over the past few years has led to the development of computation offloading decision-making algorithms, which consider various requirements and constraints to find the optimal offloading decision policy. In general, finding the optimal computation offloading decision solutions can be formulated as integer programming problems, which are NP-hard and difficult to find a globally optimal solution in real-time. To deal with the problems more tractable, various research efforts have been existed to apply well-known approximation algorithms [15]–[19] and machine learning techniques [20]–[24].

Xu *et al.* [15] formulated the deep learning offloading problem of 5G multicell MEC networks as an integer linear program (ILP), and then applied random rounding techniques to find exact and approximate solutions to ensure both efficiency and optimization with high probability. On the other hand, several researchers aimed to jointly optimize the offloading decision and resource allocation to minimize energy consumption and end-to-end latency [16]–[19]. Chen *et al.* [16] formulated a joint optimization problem as a non-convex quadratically constrained quadratic program (QCQP) and devised a heuristic algorithm by applying separable semidefinite relaxation (SDR). Liu *et al.* [17] proposed a task offloading and resource allocation framework that minimizes a user's energy consumption with ultra-reliable and low-latency communication (URLLC) constraints, including probabilistic and stochastic characteristics. In addition, they proposed a two-timescale algorithm

by applying the Lyapunov stochastic optimization and matching theory to find the optimal UE-server association, task offloading, and resource allocation. Moreover, Gao *et al.* [18] proposed a DNN inference delay prediction model to estimate the processing delays that vary depending on the DNN model partition pattern and then formulated the joint problem as a mixed-integer linear programming (MILP) problem. Meanwhile, He *et al.* [19] formulated a joint optimization problem as a mixed-integer nonlinear programming (MINLP) problem, decomposed the optimization problem into a computing resource allocation (CRA) and a DNN partition deployment (DPD), and then devised low-complexity algorithms using the Markov approximation to find the suboptimal solution in polynomial time.

In [20], Liu *et al.* proposed two decentralized data offloading algorithms based on two different game theories, including the multi-item auction (MIA) and congestion game (COG), that are applicable to a heterogeneous network consisting of cellular and Wi-Fi networks. MIA maximizes the payment of mobile operators, while COG minimizes the payment of mobile subscribers. In [21]–[24], the authors applied deep learning approaches, supervised learning and reinforcement learning to find the optimal offloading decision policy. In [21], Ali *et al.* proposed an energy-efficient deep learning-based offloading scheme (EEDOS) that trains the DNN model to find the optimal decision policy without an exhaustive decision-making process. Here, the DNN model consists of two fully connected hidden layers and is trained through an exhaustive dataset generated by a mathematical model. Similarly, in a previous study conducted by Yang *et al.*, another study [22] proposed a multi-task learning (MTL)-based offloading framework that jointly finds the offloading decision and resource allocation, which were formulated as classification and regression problems, respectively, through a single neural network model. Moreover, in [23] and [24], the authors applied a deep Q-learning approach to find the optimal offloading decision policy in a vehicular network environment where the network connectivity between vehicles and RSUs varies over time that was incurred by the vehicle's mobility.

In this paper, we formulated the problem of finding an object detection offloading decision policy as a binary combinatorial optimization problem with an unconstrained condition, which is NP-hard. Moreover, we devised a greedy-based offloading decision algorithm that iteratively finds the suboptimal solution in polynomial time unlike the previous studies that are mentioned previously [15]–[24]. To verify the effectiveness of the proposed algorithm, in this paper, we mathematically prove that the proposed greedy algorithm is able to find a globally optimal solution under certain condition.

III. PROPOSED OBJECT DETECTION OFFLOADING FRAMEWORK

In this section, we introduce an overview of the proposed object detection offloading framework that adopts an object detection-specific task-level pipeline parallelism proposed in our previous work [12] to improve the computing resource utilization on both mobile edge devices and an edge server. Afterward, we discuss preliminary experimental results that show possible problems arising in real-world object detection offloading scenarios.

A. OVERVIEW OF OBJECT DETECTION OFFLOADING APPLYING TASK-LEVEL PIPELINE PARALLELISM

Our DNN-based object detection offloading framework is designed to parallelize the workload execution by distributing heavy and lightweight workloads to the edge server and mobile edge devices simultaneously to improve the object detection performance. The proposed framework is adopted from our previous work [12], including the offloading DNN inference and NMS filtering workloads to a remote server. Figure 2 shows a block diagram that represents the proposed object detection offloading framework, which applies task-level pipeline parallelism to improve the computing resource utilization on both edge devices and a server. We categorized the object detection workloads into five tasks to apply the task-level pipeline parallelism, as follows:

- *READ*: Capture images from the camera module.
- *PRE-PROCESS*: Resize the captured image to fit the predefined input shape of the DNN model.
- *INFERENCE*: Execute data analysis based on pretrained DNN model.
- *POST-PROCESS*: Extract a set of object instances, including the bounding box location and categories prediction values, and apply NMS filter.
- *WRITE*: Render the detection results on the screen.

The detailed procedure is depicted in Fig. 3.

The proposed framework is working on the edge computing architecture consisting of three entities: the mobile edge devices, remote edge server, and wireless AP. In mobile edge devices, the image data are captured using a camera module attached to the edge device. Subsequently, it is resized and normalized to fit the predefined input shape of the object detection DNN model. Then, the DNN inference, which can be executed using two methods, is performed to detect the location and classification of the objects included in the image. First, the edge devices are capable of offloading the DNN inference tasks to the edge server, which is called DNN offloading. Second, edge devices are capable of processing the DNN inference tasks under the assumption that the edge devices are equipped with a lightweight GPU hardware.

If the edge devices are decided to execute the DNN inference, the edge devices must apply an NMS filter to extract the final detection results from the DNN inference output. On the other hand, if the edge devices are chosen to offload the DNN inference, the edge server applies the NMS filter and returns the final detection results to the edge devices. Afterward, the edge devices visualize the obtained object

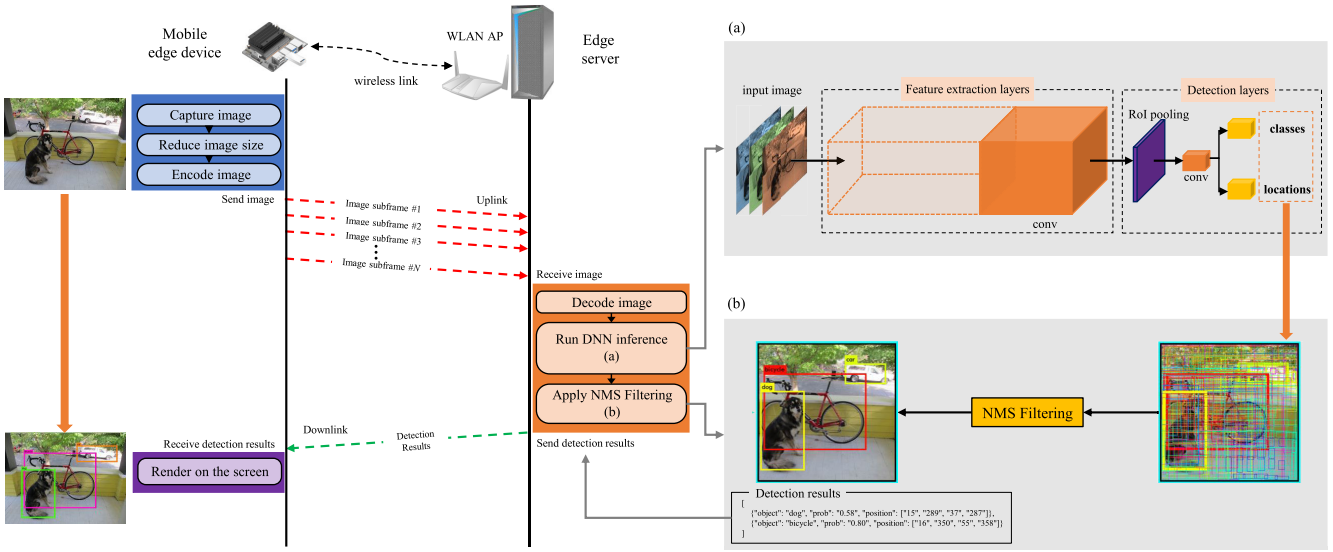


FIGURE 2. A block diagram representing the processing flow of the proposed object detection offloading framework.

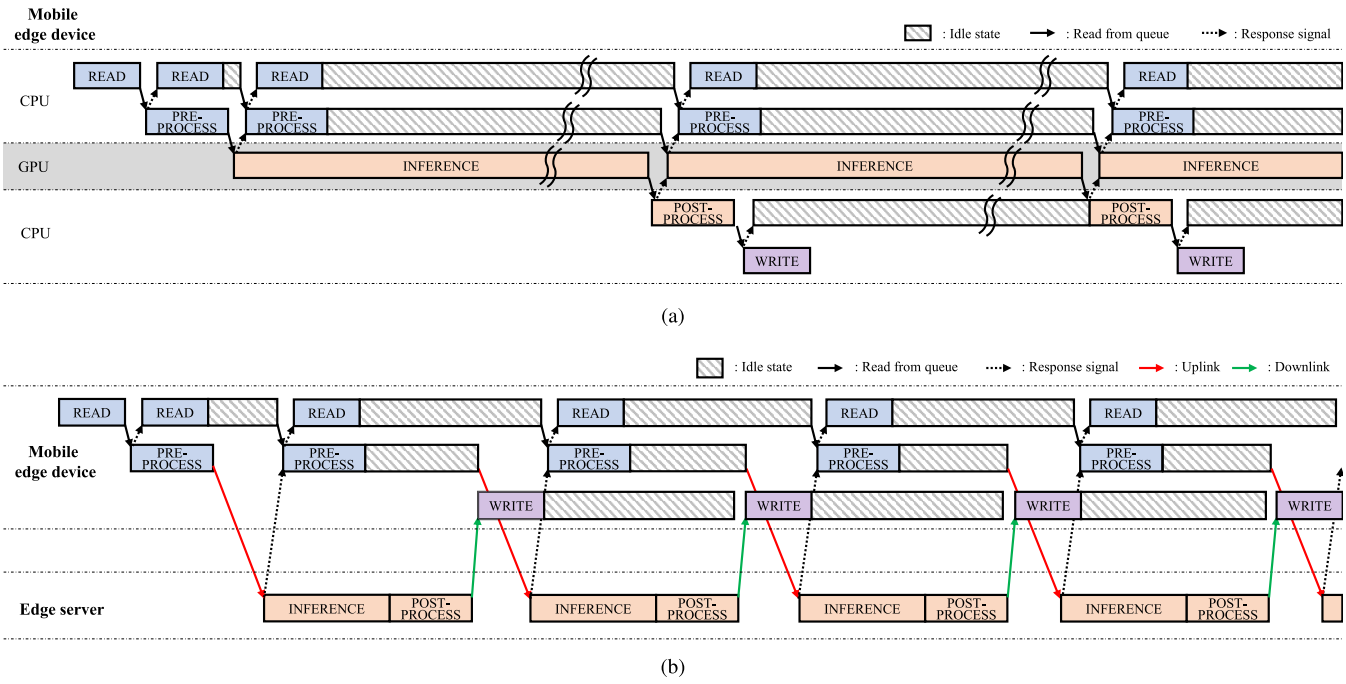


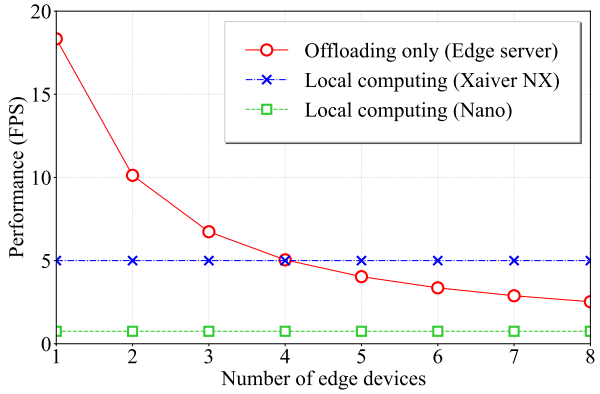
FIGURE 3. Block diagrams that represent the processing flow of the object detection that applies task-level pipeline parallelism when (a) the object detection workloads are processed locally and (b) are offloaded to the edge server.

detection results by rendering the bounding boxes and their instances to the image data.

The remote edge server is equipped with high-performance CPUs and GPUs, allowing the acceleration of the object detection tasks in comparison with edge devices that are equipped with a lightweight hardware. After pretraining the object detection DNN models, the edge server creates a DNN runtime module that executes the DNN model inference by sequentially processing micro-kernels generated by a deep learning compiler such as the Nvidia TensorRT and Apache TVM[1] [25]. Then, the edge server opens a port and waits for an object detection offloading request sent by the edge devices. Once the request arrives, the edge server executes the DNN runtime to extract the detection results. Subsequently,

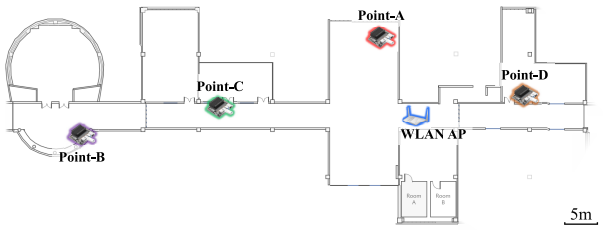
[1]Apache TVM is a domain-independent end-to-end optimizing compiler stack that transfers pre-trained deep learning models to optimized low-level codes executing on given hardware accelerators such as CPUs, GPUs, and specialized accelerators like TPUs.

(a) Object detection offloading testbed.

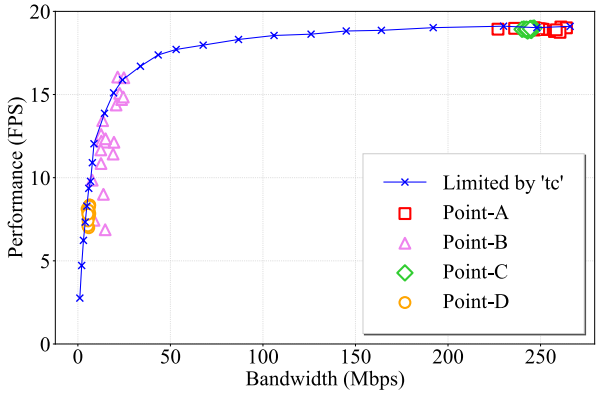


(b) Average FPS with respect to the number of mobile edge devices.

FIGURE 4. Preliminary experimental environment and results when the multiple mobile edge devices are located near the WLAN AP.



(a) Experiment scenario for the distributed mobile edge devices located on the 1st floor of ETRI Honam Research Center.



(b) Average FPS with respect to the network bandwidth and location of mobile edge devices.

FIGURE 5. Preliminary experimental environment and results when the multiple mobile edge devices are distributed over a WLAN coverage area.

the edge server applies the NMS filter to extract the final detection results, which are eventually returned to the corresponding edge device.

It is necessary to establish data communication between the mobile edge devices and the edge server in the object detection offloading framework. In particular, the latency incurred by data communication affects the detection latency of the object detection offloading method. Thus, a single-hop WLAN is appropriate for establishing data communication links to minimize the degradation. Here, the mobile edge devices are associated with a single WLAN AP where the remote edge server is attached. When the edge devices request the object detection offloading, the frames including the pre-processed image data are transmitted to the remote edge server through the uplink transmission. Afterward, the frames including the detection results are transmitted to the corresponding mobile edge devices through the downlink transmission when the detection results are delivered from the remote edge server to the edge devices.

B. PRELIMINARY EXPERIMENTAL RESULTS

We performed various preliminary experiments to evaluate the impact of various factors of the offloading performance of the object detection in terms of average FPS based on the object detection offloading framework described above. Note that the experimental setups were similar to those used in the experiments results, which is discussed in Section V and its detailed specifications are listed in Table 1.

Figure 4 shows the experimental results to evaluate the average FPS with respect to the number of mobile edge devices requesting an object detection offloading service. In this experiment, all mobile edge devices were deployed near the WLAN AP, as shown in Fig. 4(a), resulting in a similar network bandwidth to each other. As shown in Fig. 4(b), the average FPS decreases as the number of edge devices requesting object detection offloading service increases. This degradation is caused by an increase in the queueing delay, which depends on the arrival offloading request and offloading service rates, that occurs at the remote edge server. In our experiment, the service time required for object detection at the edge server was almost invariant, indicating that the queueing delay solely depends on the number of edge devices requesting an object detection offloading service. Moreover, in the case of Jetson Xavier NX, the performance of the object detection offloading to the edge server is worse than processing at the edge devices when the number of edge devices is greater than or equal to 5. The experiment results show that offloading all the workloads to the edge server results in performance degradation as the number of mobile edge devices increases, because of the limited scalability of the object detection offloading system. This emphasizes the necessity of the object detection offloading decision policy by considering the number of edge devices in order to enhance overall object detection performance in terms of processing frames per second.

Figure 5 shows the experimental results to evaluate the average FPS with respect to the network bandwidth. In this experiment, a traffic controller (tc) in the Linux kernel was used to modify the network bandwidth, while a TCPdump

library was used to measure the network bandwidth in a real-world WLAN environment. Moreover, we geographically distributed multiple mobile edge devices within the coverage area of the WLAN AP, as shown in Fig. 5(a), because the network bandwidth varies over the deployed regions of mobile edge devices. The experiment results depicted in Fig. 5(b) show that the average FPS of the object detection offloading service increases and gradually converges to the maximum value as the network bandwidth increases.[2] Here, the maximum FPS value is bounded by the number of input image FPS captured by the mobile edge device. The results show that the object detection offloading performance varies with the location of the mobile edge devices. The network bandwidth greatly depends on the signal-to-noise ratio (SNR) at the receiver, which is affected by the relative geographic location between the transmitter and receiver, such as the propagation distance and path quality (LOS vs. NLOS). This implies that the mobility of an unexpectable mobile edge device affects the object detection offloading performance because of the variation in the network bandwidth.

Through the preliminary experimental results, we found that offloading all the object detection workloads to the edge server may result in overall performance degradation of the object detection service at the mobile edge devices. In addition, the performance of the object detection offloading system is affected by the computational capacity of the edge server and the network bandwidth of the mobile edge devices. Therefore, the problem of deciding whether to offload the object detection services to the remote edge server is the most important issue for improving object detection offloading efficiency.

IV. OFFLOADING DECISION ALGORITHM

A. SYSTEM MODEL AND PROBLEM STATEMENT

Consider an object detection offloading scenario of mobile edge devices, in which the computing resources and network capacity are different. Let $N = \{n_1, \cdots, n_N\}$ denote a set of mobile edge devices in which n_i represents the ith mobile edge device. All mobile edge devices are connected to a WLAN AP to establish wireless communication links to the edge server. In this paper, we considered the close-in reference distance path loss model for the wireless signal attenuation. Then, the received signal strength at the WLAN AP transmitted from the ith mobile edge device is obtained using the following equation:

$$P_i^{\mathrm{RSS}} = 10^{\mathrm{PL}(d_0)/10} P_t d_i^{-\theta_i}, \tag{1}$$

where $\mathrm{PL}(d_0)$ is the path loss at the reference distance d_0 in the dB scale, P_t is the transmission power, and d_i and θ_i are the Euclidean distance and the path loss exponent between the ith mobile edge device and the WLAN AP, respectively.

[2]Noting that the experiment results labeled Limited by 'tc' is measured by adjusting network bandwidths through Linux kernel tc, on the other hand, the experiment results labeled Point-A∼D do not adjust their network bandwidths.

Then, the SNR at the ith mobile edge device is defined as

$$\Gamma_i = \frac{P_i^{\mathrm{RSS}}}{N_0 W}, \tag{2}$$

where N_0 is the noise power spectral density and W is the channel bandwidth. We assumed that the wireless channel is invariant over a single data transmission time, and therefore, the SNR is also invariant within a single offloading procedure.

The data rate in IEEE802.11 is closely related to the modulation and coding rates scheme (MCS) index, which is determined based on the SNR mapping method, such as exponential effective SNR mapping (EESM). The WLAN AP selects the appropriate MCS index to maximize the network capacity once the SNR is measured. Let r_i^{UL} and r_i^{DL} denote the uplink and downlink data rates of the ith mobile edge device, respectively, which are determined using the best MCS selection mechanism. Then, based on the object detection offloading procedure described in the previous section, the end-to-end latency of the object detection offloading service at the ith mobile edge device is given by

$$D_i^{\mathrm{OFF}} = \frac{M_i^{\mathrm{UL}}}{r_i^{UL}} + d_{q,i} + d_p^{\mathrm{OFF}} + \frac{M_i^{\mathrm{DL}}}{r_i^{\mathrm{DL}}}, \tag{3}$$

where M_i^{UL} and M_i^{DL} are the size of the uplink and downlink frames of the ith mobile edge device, respectively, $d_{q,i}$ is the queueing delay of the edge server at the ith mobile edge device, and d_p^{OFF} is the processing delay required to compute the object detection workload at the edge server.

The size of the uplink and downlink frames depends on the size of the input image and the number of detected objects in the image, respectively. Based on our preliminary experiments, we found that the size of the downlink frame is relatively small (i.e., $M_i^{\mathrm{UL}} \gg M_i^{\mathrm{DL}}$) than that of the uplink frame, indicating that the last term in (3) is negligible. In addition, $d_{q,i}$ can be treated as a constant based on the assumption that the object detection workloads are static and the edge server always operates at maximum performance. Therefore, the offloading delay D_i^{OFF} at the ith mobile edge device depends on the uplink data rates r_i^{UL} and the queueing delay $d_{q,i}$ incurred at the remote edge server. The queueing delay $d_{q,i}$ in (3) is obtained by the number of offloading requests stored in the queue of the edge server and the remaining time for the ongoing offloading service, which are unknown in advance. It implies that the queueing delay can be treated as a random variable with bounded support, that is, $0 \le d_{q,i} \le (N^{\mathrm{OFF}} - 1) d_p^{\mathrm{OFF}}$, in which N^{OFF} is the number of mobile edge devices offloading object detection workloads.

B. DETERMINISTIC GREEDY OFFLOADING DECISION

The objective of the object detection offloading decision problem is to maximize the average FPS of all mobile edge devices. Let $\mathbf{x} = [x_1, \cdots, x_N]$ be a binary decision vector, where $x_i = 1$ if the ith mobile edge device offloads object detection workloads to the edge server, and $x_i = 0$ otherwise.

Then, the object detection offloading decision problem is formulated as a binary optimization problem over $\mathbf{x} \in \{0, 1\}^{1\times N}$. Moreover, let β_i denote the average FPS at the ith mobile edge device when it processes object detection workloads, and $||\mathbf{x}||$ denotes a function that returns the number in a binary vector $\mathbf{x}$. Then, the globally optimal offloading decision vector $\mathbf{x}^*$ is obtained by solving the following optimization problem:

$$\mathbf{x}^* = \underset{\mathbf{x}\in\{0,1\}^{1\times N}}{\arg\max} \frac{1}{N} \sum_{i=1}^{N} \{x_i f_i(||\mathbf{x}||) + (1 - x_i)\,\beta_i\}\,, \tag{4}$$

where $f_i(\kappa)$ is the estimated average FPS of the ith mobile edge device when the κ number of devices is offloaded to the edge server. Here, the achievable average FPS by offloading the workloads greatly depends on the end-to-end object detection offloading latency and the number of input image frames captured by the edge device. Let γ_i denote the image capture FPS of the ith mobile edge device. Based on (3), $f_i(\kappa)$ is given by

$$f_i(\kappa) = \min\left\{\frac{1}{\alpha_i + \kappa \cdot d_p^{\mathrm{OFF}}}, \gamma_i\right\}, \tag{5}$$

where $\alpha_i = M_i^{UL}/r_i^{UL}$ is the time required for the uplink data transmission at the ith mobile edge device. In addition, based on (5), we applied the upper bound of the queueing delay $d_{q,i}$, that is, $(||\mathbf{x}|| - 1)d_p^{\mathrm{OFF}}$, to make the problem more tractable.

The optimal offloading decision vector in (4) can be obtained using a binary combinatorial algorithm. One may attempt to apply a brute-search method that enumerates all the possible candidates to find the best one. However, its complexity grows exponentially as the number of mobile edge devices increases. Therefore, we adopted a greedy algorithm that iteratively finds a suboptimal solution $\mathbf{x}_g^*$ by setting one element of $\mathbf{x}$ to 1. Consider a set function for the objective function $h_a(\mathcal{A}_\mathbf{x})$ as follows:

$$h_a(\mathcal{A}_\mathbf{x}) = \frac{1}{N}\left\{\sum_{i\in\mathcal{A}_\mathbf{x}} f_i(||\mathbf{x}||) + \sum_{j\in\mathbf{1}_N\setminus\mathcal{A}_\mathbf{x}} \beta_j\right\}, \tag{6}$$

where $\mathcal{A}_\mathbf{x} = \{i : x_i = 1, i = 1, \cdots, N\}$ is the index set for $x_i = 1$ in $\mathbf{x}$. Then, the algorithm selects one element $k \in \mathcal{A}_{\mathbf{1}_N} \setminus \mathcal{A}_{\mathbf{x}_g^*}$ at each iteration that maximizes the increment of the average FPS, that is, $h_a(\mathcal{A}_{\mathbf{x}_g^*} \cup \{k\}) - h_a(\mathcal{A}_{\mathbf{x}_g^*})$, where $\mathbf{1}_N$ is an all-ones vector with the size of N, until the increment is less than zero. We devised a greedy algorithm based on the following proposition:

Proposition 1: Let $g_k(c) = (\alpha_k + c)\beta_k$. For a given set of candidates $k \in \mathcal{A}_{\mathbf{1}_N} \setminus \mathcal{A}_{\mathbf{x}_g^*}$ and $c = \left(||\mathbf{x}_g^*|| + 1\right) d_p^{\mathrm{OFF}}$, a candidate k^* that minimizes the function $g_k(c)$ also maximizes the increment in the average FPS.

Proof: Appendix A ∎

We devised a greedy algorithm based on the propositions above, where the detailed procedure is described in Algorithm 1. First, $\mathbf{x}_g^* = \mathbf{0}^{1\times N}$ is set, where $\mathbf{0}^{1\times N}$ denotes the zero vector with a size of N. At each iteration, the algorithm

Algorithm 1 Proposed Offloading Decision Algorithm

1: $\mathbf{x}_g^* = \mathbf{0}^{1\times N}$;
2: $\epsilon = 0$;
3: $t = 0$;
4: **while** $\epsilon \geq 0$ **do**
5: $\quad t := t + 1$;
6: $\quad c = t \times d_p^{\mathrm{OFF}}$;
7: $\quad k^* = \arg\min_{k\in\mathcal{A}_{\mathbf{1}_N}\setminus\mathcal{A}_{\mathbf{x}_g^*}} (\alpha_i + c)\,\beta_i$;
8: $\quad \epsilon = g_a(\mathcal{A}_{\mathbf{x}_g^*} \cup \{k\}) - g_a(\mathcal{A}_{\mathbf{x}_g^*})$;
9: $\quad$ **if** $\epsilon \geq 0$ **then**
10: $\quad\quad x_g^*[k] = 1$
11: $\quad$ **end if**
12: **end while**

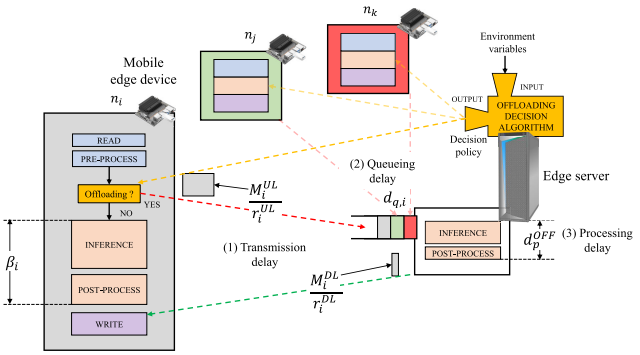


FIGURE 6. System model of the object detection offloading decision problem for supporting multiple mobile edge devices.

selects one element $k^* \in \mathcal{A}_{\mathbf{1}_N} \setminus \mathcal{A}_{\mathbf{x}_g^*}$ that maximizes the objective function in (9), as described in line 9. Subsequently, the algorithm calculates the increment of the average FPS ϵ to check whether or not to add the selected candidate $x_g^*[k] = 1$ to the optimal solution $\mathbf{x}_g^*$. Furthermore, the algorithm continues its iteration until $\epsilon < 0$, indicating that the average FPS decreases as edge devices requesting an offloading service are added to the solution $\mathbf{x}_g^*$.

Generally, the greedy algorithm may fail in finding the globally optimal solution because it is designed to find the suboptimal solution in polynomial time. The performance of this algorithm is evaluated using its derived approximation factor μ, which satisfies $h_a(\mathcal{A}_{\mathbf{x}_g^*}) \geq \mu h_a(\mathcal{A}_{\mathbf{x}^*})$, where $\mathbf{x}^*$ is the globally optimal solution. There exist various approximation algorithms that guarantee constant approximation factors when the objective function is a submodular set function. The submodular function is a set function having a nature of *diminishing returns* property, which implies that adding an element to a small set achieves more incremental value than adding the same element to a larger set. For example, given the ground set $\mathcal{N}$, a real-valued set function g is called *submodular* if it satisfies $g(\mathcal{P}\cup u) - g(\mathcal{P}) \geq g(\mathcal{Q}\cup u) - g(\mathcal{Q})$ for all $\mathcal{P} \subseteq \mathcal{Q} \subseteq \mathcal{N}$ and $u \in \mathcal{N} \setminus \mathcal{Q}$. However, the set function in (6) does not hold the submodularity condition, resulting in it being unable to derive the approximation factor of the proposed greedy algorithm described in Algorithm 1. Instead, we mathematically derived the conditions in which

TABLE 1. Summary of the experimental setup.

Component	Model	Specifications
Remote edge server	Desktop	· OS: Ubuntu 18.04.5 LTS · CPU: Intel Xeon Silver 4214 x 2 · GPU: Nvidia Titan RTX · d_p^{OFF} : 0.04
WLAN AP	NETGEAR Nighthawk RAX40	· Wireless module: Intel WAV654 chip 802.11a/n/ac/ax 2x2:2 · WLAN: IEEE802.11AC (80 MHz bandwidth & short-guard interval)
Mobile edge device	Nvidia Jetson Nano	· CPU: Quad-core ARM A57@1.43GHz · GPU: Nvidia Maxwell 128-cores · WLAN module: RTL8812BU chipset · $\beta = 0.7$
	Nvidia Jetson Xavier NX	· CPU: 6-cores Nvidia Carmel ARM@v8.2 64-bit CPU 6MB L2 + 4MB L3 · GPU: Nvidia Volta architecture with 384 CUDA cores and 48 cores · WLAN module: RTL8822CE-CG Single chip · $\beta = 5.0$
Object detection model	YOLOv3	· Input size: 608×608 · Precision: FP32 · $M^{\mathrm{UP}} = 400 \times 10^3$
DNN inference engine	Apache TVM	· Open source ML compiler · Version: 0.7dev1 · URL: https://tvm.apache.org/
Input video source	720p MP4 file	· objects: vehicles, pedestrians, bycicles, etc. · $\gamma = 30$

the proposed greedy algorithm finds a globally optimal solution based on the following proposition.

Proposition 2: Consider the arbitrary ω_i and ω_j, where $\omega_i, \omega_j \in \mathcal{A}_{\mathbf{1}_N}$ and $h_a(\{w_i\}) \geq h_a(\{w_j\})$. Then, for an arbitrary set $\Omega \subseteq \mathcal{A}_{\mathbf{1}_N} \setminus \{\{\omega_i\} \cup \{\omega_j\}\}$, the proposed greedy algorithm finds a globally optimal solution when the following condition holds:

$$h_a(\{\omega_i\} \cup \Omega) \geq h_a(\{\omega_j\} \cup \Omega). \tag{7}$$

Proof: Appendix B ■

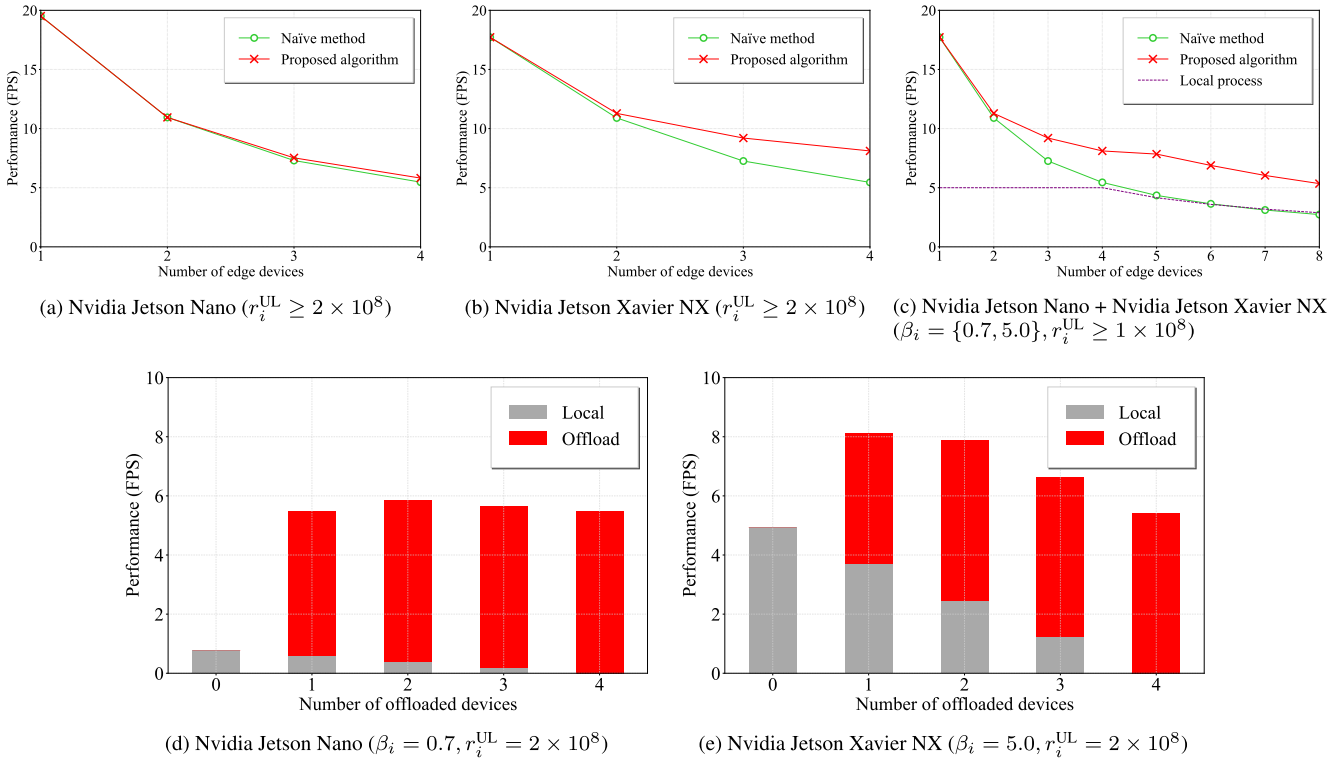
In Algorithm 1, the maximum number of iterations is equal to N. In each iteration inside the loop, the minimum from the vector of N is searched in worst case that incurs the complexity of $\mathcal{O}(N)$. Then, the worst-case runtime complexity of the proposed algorithm 1 becomes $\mathcal{O}(N^2)$, which is quadratic complexity with respect to the number of the mobile edge devices. This implies that the proposed algorithm is able to find an optimal offloading decision policy in polynomial time. Here, it is worth noting that the proposed algorithm is running on the remote edge server because it requires to gather all the environment variables measured at the remote edge devices such as their networking and computing resources as depicted in Fig. 6. In addition, the proposed offloading decision algorithm is able to be applied not only to the object detection field but also to other related fields requiring computation workload offloading services.

V. PERFORMANCE EVALUATION

In this section, we present various experimental results to evaluate the effectiveness of the proposed object detection offloading decision algorithm in real-world scenarios.

A. EXPERIMENT SETUP

We used two different types of edge devices (i.e., Nvidia Jetson Xavier NX and Nvidia Jetson Nano embedded boards) to evaluate the impact of local computing power on the offloading performance. Specifically, the Nvidia Jetson Xavier NX is equipped with more powerful computing resources, which is equipped with 6-core Nvidia Carmel ARM CPUs and 384-core Nvidia Volta GPU, than the Nvidia Jetson Nano, which has 4-core ARM Cortex-A57 CPUs and a 128-core Nvidia Maxwell GPU. On both platforms, the CUDA 10.2 library and Apache TVM runtime were installed and used to process the DNN model inference workloads locally. In addition, we used the Netgear RAX40 Wi-Fi router to provide IEEE802.11AC-based wireless connectivity operated in unlicensed 5 GHz bands to the mobile edge devices. Meanwhile, the edge server was directly attached to the router through a 1 Gbps ethernet link. Moreover, we used a desktop computer as a remote edge server equipped with Intel Xeon Silver 4214 CPU and Nvidia Titan RTX GPU. In these experiments, we used YOLOv3 (608×608, FP32) that was pre-trained using the COCO dataset as the object detection model. According to our experiments, the Nvidia Jetson Xavier NX and Jetson Nano require approximately 200 and 1,400 ms to execute the DNN model inference workloads, respectively, once the YOLOv3 model was locally processed. On the other hand, the edge server requires only approximately 40 ms to execute DNN model inference workloads. Furthermore, we considered a naïve method without an offloading decision control that allows all mobile edge devices to always offload the object detection workloads to the edge server. In the experiment, we considered two object detection offloading scenarios that depend on



(a) Nvidia Jetson Nano ($r_i^{\mathrm{UL}} \geq 2 \times 10^8$)
(b) Nvidia Jetson Xavier NX ($r_i^{\mathrm{UL}} \geq 2 \times 10^8$)
(c) Nvidia Jetson Nano + Nvidia Jetson Xavier NX ($\beta_i = \{0.7, 5.0\}, r_i^{\mathrm{UL}} \geq 1 \times 10^8$)
(d) Nvidia Jetson Nano ($\beta_i = 0.7, r_i^{\mathrm{UL}} = 2 \times 10^8$)
(e) Nvidia Jetson Xavier NX ($\beta_i = 5.0, r_i^{\mathrm{UL}} = 2 \times 10^8$)

FIGURE 7. Average FPS with respect to the number of mobile edge devices that are located near the WLAN AP as shown in Fig. 4(a).

whether the network bandwidth of the mobile edge device is balanced.

B. EXPERIMENTAL RESULTS FOR BALANCED NETWORK BANDWIDTH CASE

Figure 7 shows the experimental results that represent the average FPS with respect to the number of mobile edge devices when the network bandwidth is evenly shared among all mobile edge devices. All mobile edge devices were deployed close to the Wi-Fi AP to evenly distribute the network bandwidth, as shown in Fig. 4(a).

Figure 7(a) shows the average FPS with respect to the number of mobile edge devices, which use Nvidia Jetson Nano embedded boards. In the figure, the performances of the naïve method and the proposed algorithm gradually decrease as the number of mobile edge devices increases. The degradation that is observed in the naïve method is caused by an increase in the queueing delay incurred by the concurrent offloading decision requests, while the degradation in the proposed algorithm is caused by an increase in the number of mobile edge devices that execute object detection workloads locally. The results show that the performances of the proposed algorithm and naïve method are nearly equal to each other. The proposed object detection offloading decision algorithm is designed to maximize its detection performance by exploiting all available computing resources of the remote edge server and mobile edge devices. As a result, we found that the performance enhancement obtained by applying the proposed algorithm decreases when the local computing power of the mobile edge devices is relatively low.

On the other hand, Fig. 7(b) shows the average FPS with respect to the number of mobile edge devices, which are Nvidia Jetson Xavier NX embedded boards. The performances of both methods decrease as the number of mobile edge devices increases, which is similar to the previous results. However, the performance difference between the naïve method and the proposed algorithm increases as the number of mobile edge devices increases. An improved performance can be achieved by enabling some devices to run computationally intensive workloads locally without offloading them to the remote edge server. Specifically, the improvement of the performance of the proposed algorithm is greater than that of the previous results, which is shown in 7(a). In addition, Figs. 7(d) and (e) show the average FPS with respect to the number of offloading mobile edge devices, which are Nvidia Jetson Nano and Nvidia Jetson Xavier NX boards, respectively. It is worth noting that the computing power of the Nvidia Jetson Xavier NX is superior than that of Nvidia Jetson Nano, as shown in Fig. 4, implying that the proposed algorithm leverages more local computing capabilities to increase the detection performance. Therefore, the proposed offload algorithm leverages all available resources in the edge computing infrastructure to maximize the average FPS performance of all mobile edge devices.

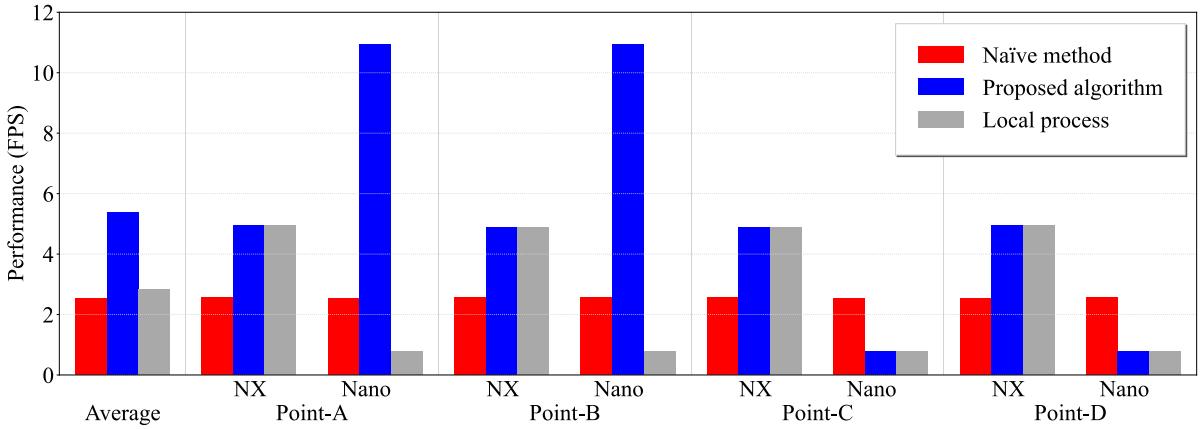


FIGURE 8. Average FPS of mobile edge devices distributed at the location marked in Fig. 5 (a).

Figure 7(c) shows the average FPS with respect to the number of mobile edge devices, consisting of Nvidia Jetson Nano and Nvidia Jetson Xavier NX boards. The figure shows that the proposed algorithm also achieves better detection performance compared with the naïve method, which is similar to the previous results shown in Fig. 7(b). This result indicates that the proposed algorithm finds the optimal offloading decision algorithm by considering the different local computing power of the mobile edge devices to maximize the overall detection performance.

C. EXPERIMENTAL RESULTS FOR IMBALANCED NETWORK BANDWIDTH CASE

We evaluated the performance of the proposed object detection offloading decision algorithm when mobile edge devices have imbalanced network bandwidths. In this experiment, we used eight mobile edge devices that were geographically distributed over the 1st floor of the ETRI Honam Research Center, as described in Fig. 5 (a). At each location, Nvidia Jetson Nano and Nvidia Jetson Xavier NX boards were deployed close together, enabling them to exploit the network bandwidth in balance. Figure 8 shows the average FPS of the mobile edge devices, and the proposed decision algorithm achieves better performance. The proposed algorithm allowed only two mobile edge devices to offload their object detection workloads, while the others were allowed to execute the workloads locally. Particularly, the algorithm selected two Nvidia Jetson Nano boards located in Point-A and Point-B with higher network bandwidth than the other positions, as shown in Fig. 5(b), indicating that offloading computationally intensive workloads of a mobile edge device with lower computing power and higher network bandwidth improves the average FPS. Therefore, the proposed object detection offloading decision algorithm makes the edge computing infrastructure fully utilize their computing and networking resources to provide the best detection performance.

VI. CONCLUSION

In this paper, we introduced a DNN-based object detection offloading framework in an edge computing infrastructure consisting of multiple mobile edge devices and a remote edge server. Through the preliminary experimental results, we verified that the proposed object detection offloading framework enhances the performance of the object detection in terms of average FPS by offloading computationally intensive workloads from mobile edge devices to the remote edge server connected through one-hop wireless links. Moreover, we found that the overall performance of the object detection service at the mobile edge devices may be degraded by offloading all the object detection workloads to the edge server because of its limited computing resources. We formulated the offloading decision problem as a binary combinatorial optimization problem and proposed an algorithm that finds the optimal solution in polynomial time. In addition, we mathematically analyzed the effectiveness of the proposed algorithm by deriving a certain condition when the algorithm always finds a globally optimal solution. Furthermore, we performed various experiments in real-life WLAN environments to verify that the proposed offloading decision algorithm maximizes the average FPS. In future work, we will extend our object detection offloading framework and offloading decision algorithm by applying them in more realistic scenarios, where multiple edge servers can provide object detection offloading services with various DNN-based object detection models. In addition, we will upgrade the proposed offloading decision algorithm to jointly optimize offloading decision and resource allocation, and then compare its performance with the other offloading decision algorithms in order to evaluate the objective performance of our algorithm.

APPENDIX A PROOF OF PROPOSITION 1

In (5), $1/\{\alpha_i + ||\mathbf{x}_g^*||d_p^{\mathrm{OFF}}\}$ decreases as the number of ones in $\mathbf{x}_g^*$ increases. Therefore, the function $f_i(||\mathbf{x}_g^*||)$ may return to

$1/\{\alpha_i + ||\mathbf{x}_g^*|| d_p^{\mathrm{OFF}}\}$ as the iteration increases. Then, the increment in the average FPS for an arbitrary edge device $k \in \mathcal{A}_{\mathbf{1}_N} \setminus \mathcal{A}_{\mathbf{x}_g^*}$ is arranged as follows:

$$\begin{aligned} & h_a(\mathcal{A}_{\mathbf{x}_g^*} \cup \{k\}) - h_a(\mathcal{A}_{\mathbf{x}_g^*}) \\ &= \frac{1}{N}\left\{ \sum_{i \in \{\mathcal{A}_{\mathbf{x}_g^*} \cup \{k\}\}} f_i(||\mathbf{x}_g^*|| + 1). + \sum_{j \in \mathcal{A}_{\mathbf{1}_N} \setminus \{\mathcal{A}_{\mathbf{x}_g^*} \cup \{k\}\}} \beta_i \right\} \\ &\quad - \frac{1}{N}\left\{ \sum_{i \in \mathcal{A}_{\mathbf{x}_g^*}} f_i(||\mathbf{x}_g^*||) + \sum_{j \in \mathcal{A}_{\mathbf{1}_N} \setminus \mathcal{A}_{\mathbf{x}_g^*}} \beta_i \right\} \\ &= \frac{1}{N}\left[\left\{ f_k(||\mathbf{x}_g^*|| + 1) - \beta_k \right\} \right. \\ &\quad \left. + \left\{ \sum_{i \in \mathcal{A}_{\mathbf{x}_g^*}} f_i(||\mathbf{x}_g^*|| + 1) - f_i(||\mathbf{x}_g^*||) \right\} \right]. \end{aligned}$$

In the equation, the first term represents the increase in the FPS at the kth device by setting $x_g^*[k] = 1$, while the second term represents the decrease in the FPS for all edge devices in $i \in \mathcal{A}_{\mathbf{x}_g^*}$. The second term remains constant regardless of the selection of k, implying that the edge device $k \in \mathcal{A}_{\mathbf{1}_N} \setminus \mathcal{A}_{\mathbf{x}_g^*}$ that maximizes the first term also maximizes the increment of the average FPS, that is, $h_a(\mathcal{A}_{\mathbf{x}_g^*} \cup \{k\})$. Then, the optimal candidate k^* is obtained by solving the following:

$$k^* = \underset{k \in \mathcal{A}_{\mathbf{1}_N} \setminus \mathcal{A}_{\mathbf{x}_g^*}}{\arg\max} \ \min \left\{ \frac{1}{\alpha_k + (||\mathbf{x}^*|| + 1)\, d_p^{\mathrm{OFF}}}, \gamma_k \right\} - \beta_k. \tag{8}$$

By substituting $c = \left(||\mathbf{x}_g^*|| + 1\right) d_p^{\mathrm{OFF}}$ and multiplying $\alpha_i + c$ to the nominator and denominator of the first term of the objective function, the optimal candidate k^* is selected by the following minimization problem:

$$k^* = \underset{k \in \mathcal{A}_{\mathbf{1}_N} \setminus \mathcal{A}_{\mathbf{x}_g^*}}{\arg\min} \ (\alpha_i + c)\, \beta_i. \tag{9}$$

APPENDIX B PROOF OF PROPOSITION 2

Suppose that $\mathbf{x}_g^*$ starts from all zero vector $\mathbf{0}_N$ such that $\mathcal{A}_{\mathbf{x}_g^*} = \emptyset$. When the ωth element of $\mathbf{x}_g^*$ is changed from 0 to 1, the variation of the objective function $h_a(\mathcal{A}_{\mathbf{x}_g^*} \cup \{\omega\}) - h_a(\mathcal{A}_{\mathbf{x}_g^*})$ is $f_\omega(1) - \beta_\omega$. This implies that, at the first iteration, the greedy algorithm described in Algorithm 1 finds ω^* maximizing the variation and changes its value from 0 to 1, i.e., $\omega^* = \arg\max_{\omega \in \mathcal{A}_{\mathbf{x}_g^*}} f_\omega(1) - \beta_\omega$. In other words, for all $\omega \in \mathcal{A}_{\mathbf{1}_N} \setminus \{\omega^*\}$, the following holds:

$$\begin{aligned} h_a(\{\omega^*\}) - h_a(\{\omega\}) &= \sum_{i \in \{\omega^*\}} f_i(1) + \sum_{j \in \mathcal{A}_{\mathbf{1}_N} \setminus \{\omega^*\}} \beta_j \\ &\quad - \left\{ \sum_{p \in \{\omega\}} f_p(1) + \sum_{q \in \mathcal{A}_{\mathbf{1}_N} \setminus \{\omega\}} \beta_q \right\} \\ &= f_{\omega^*}(1) - f_\omega(1) - \beta_{\omega^*} + \beta_\omega \geq 0. \end{aligned} \tag{10}$$

Then, for an arbitrary $\Omega \subseteq \mathcal{A}_{\mathbf{1}_N} \setminus \{\omega^*\}$ in which $||\Omega|| = \kappa$ and $\omega \in \mathcal{A}_{\mathbf{1}_N} \setminus \{\{\omega^*\} \cup \Omega\}$, the following holds:

$$\begin{aligned} & h_a(\{\omega^*\} \cup \Omega) - h_a(\{\omega\} \cup \Omega) \\ &= \sum_{i \in \{\omega^*\} \cup \Omega} f_i(\kappa + 1) + \sum_{j \in \mathcal{A}_{\mathbf{1}_N} \setminus \{\omega^* \cup \Omega\}} \beta_j, \\ &\quad - \left\{ \sum_{p \in \{\omega \cup \Omega\}} f_p(\kappa + 1) + \sum_{q \in \mathcal{A}_{\mathbf{1}_N} \setminus \{\omega \cup \Omega\}} \beta_q \right\} \\ &= f_{\omega^*}(\kappa + 1) - f_\omega(\kappa + 1) - \beta_{\omega^*} + \beta_\omega \geq 0. \end{aligned} \tag{11}$$

Let $\theta = \beta_{\omega^*} - \beta_\omega$. Then, the condition in (7) is always true when $f_{\omega^*}(\kappa+1) - f_\omega(\kappa+1) \geq \theta$ for all $\kappa = 0, \cdots, N$. Based on (5), it can be rewritten by

$$\begin{aligned} & f_{\omega^*}(\kappa + 1) - f_\omega(\kappa + 1) \\ &= \frac{\alpha_{\omega^*} - \alpha_\omega}{(\alpha_{\omega^*} + (\kappa + 1) d_p^{\mathrm{OFF}})(\alpha_\omega + (\kappa + 1) d_p^{\mathrm{OFF}})} \geq \theta \end{aligned} \tag{12}$$

Let $\eta = \alpha_{\omega^*} - \alpha_\omega$. It is worth noting that the left term of (12) increases as κ increases if $\eta < 0$. In contrast, the left term of (12) decreases as κ increases when $\eta > 0$. Therefore, if $\theta < 0$, the above condition always holds regardless of the value of η but may not hold when $\theta > 0$. By reorganizing (12), we have

$$\alpha_\omega > \frac{\theta((\kappa + 1) d_p^{\mathrm{OFF}})^2 + \alpha_{\omega^*}(1 + \theta \cdot (\kappa + 1) d_p^{\mathrm{OFF}})}{1 - \theta((\kappa + 1) d_p^{\mathrm{OFF}} + \alpha_{\omega^*})}. \tag{13}$$

The right term of the above equation increases as κ increases in the range of $\kappa = 0, \cdots, \lfloor (1 - \theta \cdot \alpha_\omega^*)/(\theta \cdot d_p^{\mathrm{OFF}}) - 1 \rfloor$ because α_{ω^*} is invariant over κ. Note that $\alpha_\omega = M_\omega^{\mathrm{UL}} / r_i^{\mathrm{UL}}$, we can derive the following condition:

$$\begin{aligned} r_\omega^{UL} &< \frac{M_\omega^{UL}(1 - \theta((\kappa + 1) d_p^{\mathrm{OFF}} + \alpha_{\omega^*}))}{\theta((\kappa + 1) d_p^{\mathrm{OFF}})^2 + \alpha_{\omega^*}(1 + \theta \cdot (\kappa + 1) d_p^{\mathrm{OFF}})}, \\ &< \frac{M_\omega^{UL}(1 - \theta(d_p^{\mathrm{OFF}} + \alpha_{\omega^*}))}{\theta(d_p^{\mathrm{OFF}})^2 + \alpha_{\omega^*}(1 + \theta \cdot d_p^{\mathrm{OFF}})} = \Psi_{\omega, \omega^*}. \end{aligned} \tag{14}$$

This implies that for an arbitrary $\omega \in \mathcal{A}_{\mathbf{1}_N} \setminus \{\Omega \cup \{\omega^*\}\}$, it is unable to hold $h_a(\{\omega\}) \leq h_a(\{\omega^*\})$ and $\beta_\omega \leq \beta_{\omega^*}$ simultaneously if its uplink transmission rate r_ω^{UL} is greater than Ψ_{ω,ω^*}. In other words, the condition in (12) always holds if there exists at least one element ω whose uplink transmission rate is greater than Ψ_{ω,ω^*}. Consequently, for a given ω^*, the greedy algorithm finds the globally optimal solution if there exists at least one element ω that satisfies $r_\omega^{UL} \geq \Psi_{\omega,\omega^*}$ and $\beta_\omega \leq \beta_{\omega^*}$.

REFERENCES

[1] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.
[2] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019.
[3] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1421–1429.

[4] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, Aug. 2019, pp. 1–16.
[5] H. Wang and J. Xie, "User preference based energy-aware mobile AR system with edge computing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 1379–1388.
[6] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.
[7] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.
[8] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.
[9] J. Shuja, S. Mustafa, R. W. Ahmad, S. A. Madani, A. Gani, and M. K. Khan, "Analysis of vector code offloading framework in heterogeneous cloud and edge architectures," *IEEE Access*, vol. 5, pp. 24542–24554, 2017.
[10] E. Ahmed, A. Ahmed, I. Yaqoob, J. Shuja, A. Gani, M. Imran, and M. Shoaib, "Bringing computation closer toward the user network: Is edge computing the solution?" *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 138–144, Nov. 2017.
[11] S. Zhou, W. Jadoon, and J. Shuja, "Machine learning-based offloading strategy for lightweight user mobile edge computing tasks," *Complexity*, vol. 2021, Jun. 2021, Art. no. 6455617.
[12] R. Kim, G. Kim, H. Kim, G. Yoon, and H. Yoo, "A method for optimizing deep learning object detection in edge computing," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2020, pp. 1164–1167.
[13] Y. Matsubara and M. Levorato, "Neural compression and filtering for edge-assisted real-time object detection in challenged networks," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, Jan. 2021, pp. 2272–2279.
[14] P. Zhou, T. Braud, A. Zavodovski, Z. Liu, X. Chen, P. Hui, and J. Kangasharju, "Edge-facilitated augmented vision in vehicle-to-everything networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12187–12201, Oct. 2020.
[15] Z. Xu, L. Zhao, W. Liang, O. F. Rana, P. Zhou, Q. Xia, W. Xu, and G. Wu, "Energy-aware inference offloading for DNN-driven applications in mobile edge clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 4, pp. 799–814, Apr. 2021.
[16] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading decision and resource allocation for multi-user multi-task mobile cloud," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.
[17] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, Jun. 2019.
[18] M. Gao, W. Cui, D. Gao, R. Shen, J. Li, and Y. Zhou, "Deep neural network task partitioning and offloading for mobile edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.
[19] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9241–9254, Oct. 2020.
[20] D. Liu, L. Khoukhi, and A. Hafid, "Decentralized data offloading for mobile cloud computing based on game theory," in *Proc. 2nd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, May 2017, pp. 20–24.
[21] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z. H. Abbas, and S. Khaf, "A deep learning approach for energy efficient computational offloading in mobile edge computing," *IEEE Access*, vol. 7, pp. 149623–149633, 2019.
[22] B. Yang, X. Cao, J. Bassey, X. Li, T. Kroecker, and L. Qian, "Computation offloading in multi-access edge computing networks: A multi-task learning approach," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.
[23] T. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.
[24] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019.
[25] *Apache TVM*. Accessed: Sep. 1, 2021. [Online]. Available: https://tvm.apache.org/

GIHA YOON received the B.S. degree in information and communications engineering from Mokpo National University, Jeollanam-do, South Korea, in 2012, and the M.S. degree in electronic computer engineering from the Graduate School, Chonnam National University, Gwangju, South Korea, in 2017. He is currently a Senior Researcher with Honam Research Center, Electronics and Telecommunications Research Institute, Gwangju. His research interests include efficient data process for edge computing devices and FPGA-based digital logic design and implementation for immediate processing systems.

GEUN-YONG KIM received the B.S. degree in electronics engineering from Kwangwoon University, Seoul, South Korea, in 2004, and the M.S. and Ph.D. degrees from the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju, South Korea, in 2006 and 2017, respectively. Since 2006, he has been with Honam Research Center, Electronics and Telecommunications Research Institute, Gwangju. His recent research interests include edge computing software framework, ML inference, and 5G networks.

HARK YOO received the B.S. degree in electrical engineering from Yonsei University, in 1998, and the M.S. and Ph.D. degrees in communications engineering from Korea Advanced Institute of Science and Technology (KAIST), in 2000 and 2005, respectively. In 2005, he joined Honam Research Center, Electronics and Telecommunications Research Institute (ETRI), Gwangju, South Korea. His research interests include medium access control for next generation optical access networks and time-sensitive networking and hardware and software platforms for edge computing nodes for the Internet of Things (IoT), industrial IoT (IIoT), and artificial intelligence of things (AIoT).

SUNG CHANG KIM received the B.S.E.E. degree from Inha University, in 1999, and the M.S.E.E. and Ph.D. degrees from Korea Advanced Institute of Technology (KAIST), South Korea, in 2002 and 2006, respectively. In 2006, he joined Honam Research Center, Electronics and Telecommunications Research Institute (ETRI), as a Principle Researcher, working on optical network technologies. He is currently the Director of the Edge Computing Application Service Research Section, ETRI. His research interests include time sensitive network algorithm and multi-access edge computing technology.

RYANGSOO KIM received the B.S. degree in information and communications from Chungnam National University, Daejeon, South Korea, in 2010, and the M.S. and Ph.D. degrees from the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea, in 2012 and 2017, respectively. He is currently a Researcher with Honam Research Center, Electronics and Telecommunications Research Institute (ETRI), Gwangju. His research interests include online learning algorithm for resource management, network protocol design and performance analysis for wired/wireless networks, and edge computing-based deep learning inference optimization.