https://doi.org/10.7840/kics.2025.50.3.460

Comparative Performance Study of Intelligent Edge Devices

Kyungwoon Lee◆°

ABSTRACT

Edge computing offers a promising solution to the latency issues inherent in centralized cloud processing, particularly for industrial Internet of Things (IIoT) applications. However, the limited computational capabilities of edge devices pose challenges to optimal artificial intelligence (AI) workload performance. This study provides a comparative performance analysis of several edge devices, focusing on evaluating the impact of hardware accelerators like graphics processing units (GPUs) on AI application processing. We employ YOLOv8, a popular object detection model, to evaluate five tasks—image classification, object detection, pose estimation, instance segmentation, and oriented bounding box detection—by measuring job completion time (JCT), GPU utilization, and memory usage. Our findings indicate that expensive high-end devices do not always provide a proportionate performance boost, with mid-range devices frequently offering comparable inference performance for less computationally demanding tasks. These results underscore the need for a careful balance between hardware specifications and application requirements to achieve efficient and cost-effective AI deployment. Additionally, we observe that multi-threading does not consistently yield performance improvements on edge devices due to Python' s Global Interpreter Lock (GIL) overhead. This limitation highlights the need for innovative solutions, such as simultaneous task management and GPU scheduling, to improve parallelism and optimize resource utilization in edge environments.

Key Words : Artificial intelligence, Edge-AI, Edge computing, Edge devices, Hardware accelerators

Ⅰ. Introduction

With the emergence of the industrial Internet of Things (IIoT)[1], Artificial Intelligence (AI) has been increasingly deployed across various sectors, such as healthcare, automotive, and smart manufacturing, to enhance service quality and productivity[2,3]. Traditionally, data generated from IoT sensors were transmitted to cloud data centers for processing, leveraging their substantial computational resources for AI model training and inference. However, this centralized approach introduces significant latency, which can be detrimental to time-sensitive applications. Edge computing[4-6] addresses this challenge by moving computations closer to the data source, thereby allowing AI inference and training to occur on edge devices near IoT sensors.

Despite the advantages of reduced latency and bandwidth usage, edge devices are typically constrained in terms of computational power, memory, and graphics processing unit (GPU) resources compared with cloud data centers[7]. This study explores the recent advancements in edge devices, particularly adopting hardware accelerators such as GPUs for fast AI application processing. First, we compare the hardware specifications of different edge devices, such as central processing unit (CPU), memory, input/output (I/O), and GPU, including the price. Thereafter, we conduct performance evaluations on edge devices using one of the most popular AI applications, YOLO. In particular, we utilize five object detection tasks of YOLOv8 and measure the job completion time (JCT)

※ This research was supported by Kyungpook National University Research Fund, 2022.
◆° First and Corresponding Author : Kyungpook National University, School of Electronics Engineering, kwlee87@knu.ac.kr, 정회원
논문번호 : 202409-209-C-RE, Received September 13, 2024; Revised November 7, 2024; Accepted November 11, 2024

and resource usage, such as GPU and memory usage.

Our evaluation results demonstrate that devices with expensive hardware specifications do not always offer a comparatively high performance. Devices with moderate hardware specifications also provide a high inference performance for low-computation jobs. For instance, the performance gap between two devices is only 1% despite twice the difference in GPU cores. In addition, we find that multi-threading does not consistently yield performance improvements on edge devices because of the global interpreter lock that prevents concurrent execution of multiple threads.

This study aims to provide a comparative analysis of the performance of various edge devices in AI inference tasks, focusing on the trade-offs between hardware specifications and application performance. By benchmarking devices with different CPU, GPU, and memory configurations, we explore the potential of moderately priced edge devices to deliver competitive performance for certain AI workloads. The results offer valuable insights into selecting edge devices based on specific computational requirements and cost considerations, ultimately contributing to more efficient deployment of AI applications in edge environments.

Ⅱ. Background and motivation

This section presents the hardware specifications of the recently developed edge devices that use hardware accelerators to achieve high-performance inference processing. In addition, we review recent studies that have focused on inference processing in edge environments.

2.1 Edge devices and hardware accelerators

Since the advent of edge computing, several research groups have adopted Raspberry Pi for implementation and experimentation[5,8-10]. Raspberry Pi is a single-board computer with an ARM processor, memory, networking capabilities, storage, and other components. Its ability to run general-purpose operating systems such as Linux allows legacy applications to be executed on the Raspberry Pi without modification. However, compared to the servers used in cloud data centers, the hardware specifications of Raspberry Pi are relatively limited. This is particularly evident in tasks involving deep learning (DL) models, such as inference or training, which require substantial computational power and take considerable time to complete[11].

To accelerate DL tasks on edge devices, vendors have introduced new types of hardware equipped with specialized accelerators. For instance, Google developed the Coral AI board, a single-board computer featuring a tensor-processing unit for faster inference[12]. Similarly, Intel offers neural computing ticks that leverage vision processing units to accelerate computer vision workloads[13]. NVIDIA provides the most extensive range of edge computing products, including GPUs and toolkits, with offerings such as Jetson Nano, Orin Nano, and AGX Orin (as listed in Table 1). These are only a few examples of NVIDIA' s comprehensive Jetson product line[14].

Table 1 summarizes a detailed comparison of the hardware specifications for various NVIDIA Jetson products, highlighting key components such as the CPU, memory, I/O, and GPU. The Jetson Nano is an entry-level device designed for low-power edge AI applications, which can be ideal for lightweight AI workloads, such as basic computer vision tasks. Orin Nano is a more advanced edge device with a significantly enhanced ability to handle complex AI workloads, such as real-time inference for robotics and autonomous systems. At the high end, the AGX Orin boasts a 2048core (at maximum) NVIDIA Ampere GPU, which provides server-grade performance for demanding AI applications and becomes suitable for industrial and research applications requiring high performance and energy efficiency.

The key differences between these devices are in the number of CPU and GPU cores, memory capacity, and the GPU architecture used. The Jetson Nano is based on the NVIDIA Maxwell architecture and is designed to enhance power efficiency. By contrast, Orin Nano and AGX Orin use a more advanced NVIDIA Ampere architecture, which delivers higher performance and efficiency through dedicated tensor cores and a significantly larger memory bandwidth. These variations in the hardware specifications result in considerable pricing differences. For instance, the

Table 1. Popular NVIDIA Jetson products and the hardware specifications, which are utilized for Edge-AI.

Device model	Jetson Nano	Orin Nano	AGX Orin
CPU	ARM Cortex-A57@1.43 GHz	ARM Cortex®-A78AE@1.5 GHz	ARM Cortex-A78AE@2.2 GHz
# of cores	4	6	12
Memory	2/4GB	4/8GB	32/64GB
Network	1GbE		Up to 10GbE
Storage	microSD card/16GB eMMC 5.1	Supports external NVMe	64GB eMMC 5.1
GPU	128 CUDA cores	512/1024-core GPU with 16/32 Tensor cores	1792/2048-core GPU with 56/64 Tensor cores
GPU architecture	NVIDIA Maxwell	NVIDIA Ampere	

AGX Orin is the most expensive device in the Jetson lineup and offers the highest memory capacity and server-grade CPU and GPU capabilities. In contrast, the Jetson Nano is far more affordable, costing approximately 1/20th the price of the AGX Orin but with fewer CPU and GPU cores and a smaller memory capacity.

2.2 Recent studies on Edge-AI

Although edge devices utilize hardware accelerators to accelerate AI application processing, they still experience performance degradation compared to powerful cloud servers because of their limited memory capacity and fewer GPU cores. Significant research efforts have focused on optimizing the performance of edge devices to address these performance limitations while maintaining the benefits of edge computing. They can be categorized into 1) task offloading, 2) model partitioning and distribution, and 3) system optimization.

1) Task offloading (*to cloud or hardware*) to maximize the inference performance of DL models on edge computing devices and hardware accelerators such as GPUs are utilized[15,16]. Traditional edge devices, such as Raspberry Pi, do not support GPUs and only include CPUs and memory, resulting in long processing times for DL inference, which requires significant computation. However, NVIDIA and Google have recently released edge devices with hardware accelerators such as GPUs and NPUs. These hardware accelerators enable real-time inference with fast computational performance, thereby enhancing the usability of DL models in various edge applications[17].

In addition, studies on task offloading techniques that distribute computational tasks from edge devices to the cloud or high-performance hardware are actively underway[18]. This helps overcome the resource limitations of edge devices and maximizes the performance of DL models by leveraging the powerful computing capabilities of the cloud. In particular, dynamic offloading techniques that determine the optimal offloading strategies by considering real-time changes in network conditions and resource availability have been extensively studied[19,20].

2) Lower the burden on DL models by partitioning or distributing studies is also actively underway to enhance computational performance on edge devices by reducing the computational load required by DL models themselves. These include distributed inference methodology, which distributes computations across multiple edge devices[17,21,22]. For instance, model partitioning divides a DL model into several parts, each computed using different edge devices. This approach improves overall system performance by reducing memory usage[11], increasing computation speed, or minimizing energy consumption[21].

3) System optimization is another key research area aimed at maximizing DL inference performance in edge computing environments. This focuses on developing resource management and scheduling algorithms to optimize resource usage in edge devices[23-25]. This is essential for satisfying the requirements of real-time applications and ensuring that inference tasks are performed within a specified time-frame to achieve the target performance[26]. For instance, upon receiving an inference request from a user, an appropriate edge device is selected to perform

the task, and the necessary computing resources, such as a GPU or memory, are allocated to achieve the desired performance. This efficient utilization of the limited computing resources on edge devices ensures that the service quality demanded by users is satisfied.

Ⅲ. Performance evaluation

This section evaluates and compares the AI application performance using the three Edge-AI devices. The following subsections describe the experiment setting and the results.

3.1 Experiment method

We evaluated the application performance of various edge devices to analyze the impact of the hardware specifications on the performance. YOLOv8[27], a widely used object detection application, was employed for the tests. Specifically, we executed five distinct tasks provided by YOLOv8 and measured the JCT for processing 100 images. The five tasks included image classification, object detection, pose estimation, instance segmentation, and oriented bounding box (OBB) object detection. Image classification assigns an image to one of the predefined classes, whereas object detection identifies the location and class of objects within an image. Pose estimation pinpoints the key points in an image, whereas instance segmentation isolates individual objects by generating masks that separate them from the background. OBB object detection extends standard object detection by introducing angular data, enabling more precise object localization.

For each task, we utilize pre-trained YOLOv8 models without performing additional training, data preprocessing, or hyperparameter tuning, allowing us to concentrate exclusively on inference performance. The models are provided in TensorRT[28] format, which optimizes them for use on NVIDIA GPUs, enabling fast and efficient inference. Different datasets are employed for each task and aligned with the pre-training dataset of each model. For example, the model for OBB is pre-trained on the DOTA[29] dataset, the classification model on ImageNet[30], and other models on the COCO[31] dataset. For each task, we randomly select 100 images from the respective dataset for evaluation. Table 2 presents the average and standard deviation of image sizes based on the dataset.

Table 2. Average and standard deviation of the image sizes depending on the datasets in KB.

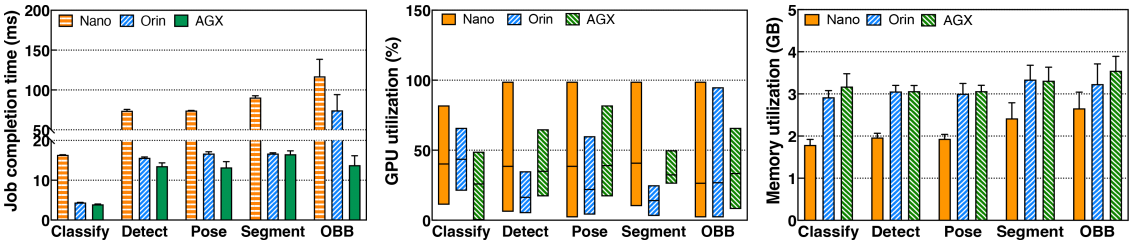
	COCO	DOTA	ImageNet
AVG	147.9	6243.9	40.5
STD	53.1	8522.9	15.3

In addition to the JCT, we measured GPU utilization and memory usage over time using jetson-stats[32] as the tasks were executed on edge devices. All tasks were run within containerized environments[1] using Nvidia-docker (version 25.0.3) to ensure consistent execution environments, such as matching versions of Python libraries. However, the Linux kernel versions varied between devices because NVIDIA no longer supports Jetson Nano beyond Jetpack 4.6.1. Consequently, Jetson Nano operates on kernel version 4.9.253, whereas Orin Nano and AGX Orin use version 5.10.120.

3.2 Individual workload performance

Fig. 1a shows the average JCT and the standard deviation for different tasks and devices where the JCT indicates the time for inferencing an image. The JCT increases depending on the computational complexity of the tasks. For instance, image classification (i.e., *Classify*) achieves the shortest JCT because it is the most straightforward job among the five tasks. In contrast, instance segmentation (i.e., *Segment*) and *OBB* require relatively long JCT for complex tasks, such as masking and locating. When we compare the JCT of the devices, Jetson Nano (i.e., Nano) has the longest JCT, followed by AGX Orin (i.e., AGX) and Orin Nano (i.e., Orin). This is because the number of GPU cores differs depending on the device, as listed in Table 1. We measure the GPU utilization while executing each task. As shown in Fig. 1b, Nano utilizes almost all of the GPU cores except for Classify. This differs from AGX and Orin, which utilize 60% of GPU cores at maximum. Note that GPU utilization varies over time, and we present the minimum and

1) We utilize a container image, ultralytics:latest-jetson.



(a) JCT per image. (b) GPU utilization. (c) Memory usage.

Fig. 1. Using Jetson Nano (Nano), Orin Nano (Orin), and AGX Orin (AGX), we evaluate the JCT of five tasks provided by YOLOv8 while measuring resource usage simultaneously.

maximum GPU utilization of each device when executing tasks.

In addition, Orin offers a high performance similar to AGX except for *OBB*, despite the difference in the number of GPU cores. For instance, *Segment* achieves almost the same average JCT as AGX and Orin, where the gap is only 0.2*ms*. This shows that Orin can be a better option for achieving a low inference latency for lightweight inference jobs than AGX, which costs four times. Moreover, Fig. 1c illustrates that all tasks do not fully utilize the memory capacity of the devices, which is less than 4*GB*. For Nano, the memory usage of each task becomes limited by the saturated GPU cores and does not exceed 3*GB*. We can assume that the impact of memory capacity is lower than that of the number of GPU cores.

For *OBB*, the most computation-intensive job among the five tasks, AGX significantly outperforms the other devices because of the large number of GPU cores. AGX offers fast JCT by 5.4 × and 8.5 × compared to Orin and Nano. However, this is the largest performance improvement from running AGX compared to Orin and Nano. The performance gains from AGX compared with Orin and Nano for the other four tasks were only 1.1 × and 5.2 ×, respectively, on average. This can lower the user's expectation of high performance when selecting AGX, considering the high price of the device, which is more expensive than Orin and Nano by 5 × and 20 ×.

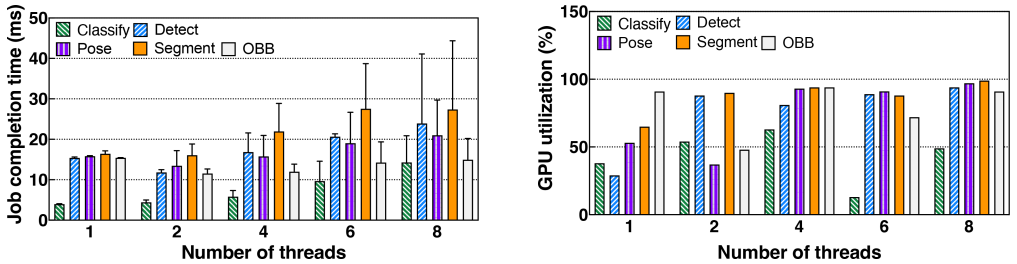
3.3 Multi-threading performance

Next, we evaluate the five YOLOv8 tasks by increasing the number of threads to maximize performance. This experiment is conducted on AGX to take full advantage of its superior hardware capacity compared to other devices while maintaining the same experimental setup described in Section 3.2. For the experiment, we create multiple threads, ranging from one to eight, using Python's **Thread** method, allowing the threads to perform the same task on different sets of images simultaneously. We then calculate the average job completion time (JCT) per image and monitor overall GPU utilization. It is important to note that we use **Thread** instead of **Process** because multi-processing requires separate memory allocation for each task, which is inefficient for edge devices.

When measuring GPU utilization (Fig. 2b), we observe that utilization does not consistently increase with a higher number of threads. The limited parallelism on AGX is primarily due to Python's Global Interpreter Lock (GIL), which ensures that only one thread executes Python bytecode simultaneously. Because the GIL prevents the concurrent execution of multiple YOLOv8 threads, each thread must wait for the completion of another, thereby limiting performance gains from running multiple threads concurrently. This constraint significantly hampers the performance improvements achievable with YOLOv8 through multithreaded execution.

Ⅳ. Implication

Fig. 1 depicts the application performance and resource usage depending on the task and devices. From the experimental results, flagship devices, such as AGX do not always offer outstanding performance in image inferencing, such as YOLOv8. For inference



(a) JCT per image. (b) GPU utilization.

Fig. 2. Average inference time and GPU utilization with the increasing number of threads.

tasks with low computational overhead, cheaper devices, such as Orin, exhibit almost similar performance to AGX. From this investigation, we can conclude that job placement, considering the computational overhead and hardware specifications (i.e., the number of GPU cores) for inferring jobs in edge computing, is necessary for efficient and high-performance inferencing. However, recent studies in Section 2.2 have mostly focused on offloading and lowering the computation overhead. Other studies[17,24], such as job scheduling on heterogeneous devices, do not consider the efficiency between performance gain and device prices. For instance, Zeng et al. proposed CoEdge[17], which orchestrates cooperative inference jobs over heterogeneous edge devices. However, CoEdge aims to minimize the energy consumption within the latency requirement without considering price efficiency. Similarly, Liang et. al introduces a resource management technique for AI applications in edge environments. Although they increase resource utilization by 2.3×, heterogeneous hardware specifications for edge devices are not considered.

In addition, Fig. 2 demonstrates that multi-threading does not always lead to performance improvements on edge devices due to GIL overhead. To leverage multithreaded inference without being constrained by the GIL, application developers must manually manage job queues to submit multiple tasks to the GPU simultaneously, thus minimizing the impact of the GIL. Additionally, we observed that running multiple containers concurrently on edge devices is not feasible because these devices lack advanced functionalities like multiprocess services and multi-instance GPUs. This limitation hinders efficient GPU utilization when multiple tasks need to be executed simultaneously. Consequently, we believe that developing new system-level techniques—such as simultaneous task management, incorporating GPU scheduling and GPU virtualization—is crucial for optimizing the performance of legacy applications on edge devices.

V. Conclusions

In this study, we comprehensively evaluated various edge devices to assess their ability to handle AI workloads, particularly object detection tasks using YOLOv8. Our findings demonstrate that flagship devices, such as AGX Orin offer superior performance owing to their advanced GPU architecture and more significant core counts. However, more affordable devices, such as Orin Nano, can achieve comparable performance for tasks that are less computationally demanding. This underscores the importance of carefully selecting edge devices running AI applications based on specific requirements to efficiently balance cost and performance. Furthermore, the results highlight the limitations of current edge devices in handling computation-intensive tasks, such as OBB detection, where high-performance devices, such as AGX Orin, are necessary. As techniques for edge computing continue to evolve, future studies should focus on optimizing resource management, task scheduling, and the deployment of edge-AI systems to fully exploit the potential of hardware accelerators, thereby overcoming the inherent resource constraints of edge environments.

References

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Trans. Industrial Inf.*, vol. 14, no. 11, pp. 4724-4734, 2018. (https://doi.org/10.1109/TII.2018.2852491)

[2] D. Saraswat, P. Bhattacharya, A. Verma, et al., "Explainable Ai for healthcare 5.0: Opportunities and challenges," *IEEE Access*, vol. 10, pp. 84486-84517, 2022. (https://doi.org/10.1109/ACCESS.2022.3197671)

[3] M. A. Rahman, M. S. Hossain, A. J. Showail, N. A. Alrajeh, and A. Ghoneim, "Ai-enabled IIot for live smart city event monitoring," *IEEE Internet of Things J.*, vol. 10, no. 4, pp. 2872-2880, 2021. (https://doi.org/10.1109/JIOT.2021.3109435)

[4] W. Yu, F. Liang, X. He, et al., "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900-6919, 2017. (https://doi.org/10.1109/ACCESS.2017.2778504)

[5] C.-H. Hong, K. Lee, M. Kang, and C. Yoo, "Qcon: Qos-aware network resource management for fog computing," *Sensors*, vol. 18, no. 10, p. 3444, 2018. (https://doi.org/10.3390/s18103444)

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things J.*, vol. 3, no. 5, pp. 637-646, 2016. (https://doi.org/10.1109/JIOT.2016.2579198)

[7] T. Hao, K. Hwang, J. Zhan, Y. Li, and Y. Cao, "Scenario-based ai benchmark evaluation of distributed cloud/edge computing systems," *IEEE Trans. Comput.*, vol. 72, no. 3, pp. 719-731, 2022. (https://doi.org/10.1109/TC.2022.3176803)

[8] J. Byeon, S. Kim, H. Lee, B. Baek, and K. Lee, "Exploring kubernetes-based network performance control in edge computing environments," *J. Korean Inst. of Next Generation Comput.*, vol. 20, no. 1, pp. 114-124, 2024. (https://doi.org/10.23019/kingpc.20.1.202402.009)

[9] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Comput. Sys.*, vol. 97, pp. 219-235, 2019. (https://doi.org/10.1016/j.future.2019.02.050)

[10] B. Varghese, N. Wang, D. Bermbach, et al., "A survey on edge performance benchmarking," *ACM Comput. Surv.* (*CSUR*), vol. 54, no. 3, pp. 1-33, 2021. (https://doi.org/10.1145/3444692)

[11] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained Iot edge clusters," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Syst.*, vol. 37, no. 11, pp. 2348-2359, 2018. (https://doi.org/10.1109/TCAD.2018.2858384)

[12] *Coral: Build beneficial and privacy preserving Ai*, https://coral.ai/(Accessed on 01/09/2024).

[13] *Intel neural compute stick 2* (*intel ncs2*), https://www.intel.com/content/www/us/en/ developer/articles/tool/neuralcomputestick.html(Accessed on 01/09/2024).

[14] N. Developer, *Jetson embedded ai computing platform*, https://docs.ultralytics.com/ko/models/yolov8/(Accessed on 01/09/2024).

[15] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "Toffee: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1634-1644, 2019. (https://doi.org/10.1109/TCC.2019.2923692)

[16] Y. Wang, X. Tao, X. Zhang, P. Zhang, and Y. T. Hou, "Cooperative task offloading in three-tier mobile computing networks: An admm framework," *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 2763-2776, 2019. (https://doi.org/10.1109/TVT.2019.2892176)

[17] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM*

Trans. Netw., vol. 29, no. 2, pp. 595-608, 2020. (https://doi.org/10.1109/TNET.2020.3042320)

[18] A. Yaqoob, T. Bi, and G.-M. Muntean, "A survey on adaptive 360 video streaming: Solutions, challenges and opportunities," *IEEE Commun. Surv. & Tuts.*, vol. 22, no. 4, pp. 2801-2838, 2020. (https://doi.org/10.1109/COMST.2020.3006999)

[19] Y. Chen, N. Zhang, Y. Zhang, and X. Chen, "Dynamic computation offloading in edge computing for internet of things," *IEEE Internet of Things J.*, vol. 6, no. 3, pp. 4242-4251, 2018. (https://doi.org/10.1109/ JIOT.2018.2875715)

[20] K. Liu, C. Liu, G. Yan, V. C. Lee, and J. Cao, "Accelerating dnn inference with reliability guarantee in vehicular edge computing," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 3238-3253, 2023. (https://doi.org/10.1109/TNET.2023.3279512)

[21] J. Wu, L. Wang, Q. Pei, X. Cui, F. Liu, and T. Yang, "Hitdl: High-throughput deep learning inference at the hybrid mobile edge," *IEEE Trans. Parall. and Distrib. Syst.*, vol. 33, no. 12, pp. 4499-4514, 2022. (https://doi.org/10.1109/TPDS.2022.3195664)

[22] C. Dong, S. Hu, X. Chen, and W. Wen, "Joint optimization with dnn partitioning and resource allocation in mobile edge computing," *IEEE Trans. Netw. and Service Manag.*, vol. 18, no. 4, pp. 3973-3986, 2021. (https://doi.org/10.1109/TNSM.2021.3116665)

[23] Z. Xia, Y. Hao, J. Duan, C. Wang, and J. Jiang, "Towards optimal preemptive gpu time-sharing for edge model serving," in *Proc. 9th Int. Wkshp. on Container Technol. and Container Clouds*, pp. 13-18, 2023. (https://doi.org/10.1145/3631311.3632401)

[24] Q. Liang, W. A. Hanafy, A. Ali-Eldin, and P. Shenoy, "Model-driven cluster resource management for ai workloads in edge clouds," *ACM Trans. Autonomous and Adaptive Syst.*, vol. 18, no. 1, pp. 1-26, 2023. (https://doi.org/10.1145/3582080)

[25] A. Dhakal, S. G. Kulkarni, and K. Ramakrishnan, "Gslice: Controlled spatial sharing of gpus for a scalable inference platform," in *Proc. 11th ACM Symp. Cloud Comput.*, pp. 492-506, 2020. (https://doi.org/10.1145/3419111.3421284)

[26] H. Sun, Y. Yu, K. Sha, and H. Zhong, "Edgeeye: A data-driven approach for optimal deployment of edge video analytics," *IEEE Internet of Things J.*, vol. 9, no. 19, pp. 19273-19295, 2022. (https://doi.org/10.1109/JIOT.2022.3166896)

[27] Ultralytics, *Ultralytics yolov8*, https://developer.nvidia.com/embeddedcomputing(Accessed on 05/11/2024).

[28] N. Developer, *Nvidia tensorrt*, https://developer.nvidia.com/kokr/tensorrt(Accessed on 05/11/2024).

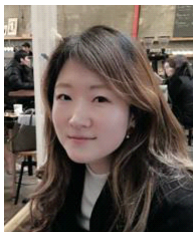
[29] Ultralytics, *Dota dataset with obb*, https://docs.ultralytics.com/datasets/obb/dota-v2/(Accessed on 05/11/2024).

[30] S. V. Lab, *Imagenet*, https://www.imagenet.org/(Accessed on 05/11/2024).

[31] Ultralytics, *Coco dataset*, https://docs.ultralytics.com/datasets/detect/coco/(Accessed on 05/11/2024).

[32] *Jetson-stats*, https://github.com/rbonghi/jetson_stats(Accessed on 05/11/2024).

Kyungwoon Lee



She received the B.E. degree from the School of Electronics Engineering, Kyungpook National University, Daegu, South Korea, and the M.S. and Ph.D. degree in computer science from Korea University, Seoul, South Korea. From 2020 to 2022, she was a research professor at the Department of Computer Science and Engineering at Korea University. She is currently working as an assistant professor in the School of Electronics Engineering, Kyungpook National University. Her research interests include resource scheduling in cloud computing, container and server virtualization, and TCP/IP kernel networking stack.