Hindawi
Mobile Information Systems
Volume 2021, Article ID 6650053, 8 pages
https://doi.org/10.1155/2021/6650053

Research Article

Energy Efficient and Real-Time Remote Sensing in AI-Powered Drone

Bongjae Kim,[1] **Jinman Jung**,[2] **Hong Min**,[3] **and Junyoung Heo**[4]

[1]*Department of Computer Engineering, Chungbuk National University, Cheongju 28644, Republic of Korea*
[2]*Department of Computer Engineering, Inha University, Incheon 22212, Republic of Korea*
[3]*School of Computing, Gachon University, Seongnam 13120, Republic of Korea*
[4]*Division of Computer Engineering, Hansung University, Seoul 02876, Republic of Korea*

Correspondence should be addressed to Jinman Jung; jmjung@inha.ac.kr and Hong Min; hmin@gachon.ac.kr

Received 24 December 2020; Revised 9 March 2021; Accepted 23 March 2021; Published 1 April 2021

Academic Editor: Hoon Ko

Remote sensing using drones has the advantage of being able to quickly monitor large areas such as rivers, oceans, mountains, and urban areas. In the case of applications dealing with large sensing data, it is not possible to send data from a drone to the server online, so it must be copied to the server offline after the end of the flight. However, online transmission is essential for applications that require real-time data analysis. The existing computation offloading scheme enables online transmission by processing large amounts of data in a drone and transferring it to the server, but without consideration for real-time constraints. We propose a novel computation offloading scheme which considers real-time constraints while minimizing the energy consumption of drones. Experimental results showed that the proposed scheme satisfied real-time constraints compared to the existing computation offloading scheme. Furthermore, the proposed technique showed that real-time constraints were satisfied even in situations where delays occurred on the server due to the processing of requests from multiple drones.

1. Introduction

Remote sensing has the advantage of being able to monitor a wide range of areas over long distances. In large areas such as rivers, oceans, mountains, and urban areas, remote sensing drones can collect a lot of data in a short period of time. These data are copied to a high-performance computer to obtain the desired information through analysis algorithms. Due to the mobility of drones and the advantage of remote sensing, the development of remote sensing applications using drones is increasing [1–3].

Sensors that are commonly used for remote sensing include image sensors, hyperspectral image sensors, and lidar. Image sensors are also widely used in hobby drones, and data transmission is sufficiently affordable with current network technology. However, in the case of hyperspectral image sensors and lidar, the size of sensing data can be more than several gigabytes per minute, so real-time transmission is very difficult. It is common for drones to store sensing data in the drone's storage device when flying and then copy the stored data to the computer for analysis after the flight is completed. Such offline analysis cannot satisfy the requirements of applications that require real-time information.

To satisfy real-time information requirements, we can think of how to obtain information by analyzing sensing data inside the drones. However, analyzing large-scale sensing data requires a high-performance computer, which is challenging to fit into drones because of its heavyweight and energy consumption. Because drones fly, they are very sensitive to load weight and battery capacity.

Considering the energy consumption and load weight, the performance of CPUs or NPUs available in embedded devices such as drones is much lower than that of high-performance servers with GPUs. Therefore, the sensing data analysis algorithm performed by the server cannot be used in drones as it is. These problems are more severe because the analysis algorithms are based on deep learning. To address this, the computation offloading scheme has been proposed

to perform some of the analytical algorithms on drones and perform the remaining algorithms after transferring the intermediate results to the server [4].

In drone-based remote sensing applications with real-time constraints, computational offloading can be a good solution. The existing computational offloading finds an optimal layer that minimizes the sum of the energy for performing deep learning-based analysis algorithms up to a specific layer and the energy for transferring intermediate result data to the server. However, this does not consider the real-time constraints at all, which may result in exceeding the deadline during computation up to the optimal layer or during computation on the server. When a GPU server handles requests from multiple drones, the latency on a GPU server often leads to an inability to satisfy the deadline.

In this paper, we propose a novel computation offloading considering the real-time constraints and delays in the server. The delay can be caused by requests from multiple drones to the server. The proposed technique calculates the time to perform deep learning-based analysis algorithms to the specific layer, the time to transmit intermediate result data to the server, and the time to wait on the server. Then, it calculates the energy consumed by performing the analysis algorithm on drones and the energy consumption required to transmit the intermediate result data to the server. Using these calculations, we find the optimal layer to minimize energy consumption while meeting the deadline.

Experimental results show that the proposed scheme, compared to the existing techniques, satisfies the real-time constraints while minimizing energy consumption. It also shows that the proposed scheme satisfies the real-time constraints even when multiple drones send requests to the server.

This paper is organized as follows: Section 2 describes the related works. Section 3 shows the proposed computation offloading. The experimental environments and results are described in Section 4, and the concluding remarks are given in Section 5.

2. Related Works

2.1. Offloading Schemes for Mobile Devices. Wu and Wolter estimated the energy consumption and delay under two delayed offloading policies, partial offloading and full offloading, and two types of network infrastructures, Wi-Fi and cellular [5]. The authors found some meaningful results. The full offloading is best if the deadline is extremely long, but partial offloading and Wi-Fi transmission are considered if the deadline is short. To reduce energy consumption, a mobile device commits its tasks as much as possible to the cloud and uses the cellular network.

Kim et al. proposed a dynamic computation offloading scheme for tracking a flying object [6]. The authors consider the mobility of the frying target and the communication failure rate between a drone and the base station to reduce object detection and controlling decision time.

Edge computing-based schemes have been studied to reduce the network delay caused by physical distance and to support real-time applications. Performance guaranteed computation offloading scheme was proposed to minimize the total energy consumption of mobile devices in mobile-edge computing [7]. In this scheme, the offloading is decided by the energy condition of each mobile device, the computational capacity of each edge server, and channel bandwidth.

Moussa et al. proposed a task offloading strategy based on load balancing for mobile-edge computing environments [8]. The authors designed an offloading model based on M/M/1 queuing system and 5G network features, for instance, orthogonal subchannels. Their optimization scheme finds a minimum delay of offloading by using several iteration searches.

Mobile-edge computation offloading was devised by Gou et al. for ultradense IoT networks [9]. The authors proposed a two-tier game-theoretic greedy offloading scheme to improve the single-tier offloading. The resource capacity of edge servers changes dramatically because a large number of tasks arrive randomly to edge servers in ultra-dense IoT networks. The limitation of wireless channels is also considered to reduce the overall computation overhead of all tasks. The two-tier model includes outer and inner layers. The outer layer is used for finding a globally optimal offloading decision, and the inner layer is used for finding a locally optimal offloading strategy.

2.2. Offloading Schemes for Deep Learning. There are many studies that use deep learning frameworks to decide on offloading for saving energy consumption of mobile devices and reducing task completion time [10–15]. However, we have a different view that is selectively offloading as the input size of each layer and finding an optimal partitioning point.

Neurosurgeon [16] is a hybrid approach between cloud and mobile-only-based deep learning framework. In this scheme, the data and layers are partitioned and committed some data and layers to the cloud for reducing energy consumption and computational latency. The authors found that the optimal partitioning point is changed according to deep learning models, hardware platforms, server load levels, and wireless channel bandwidth.

A Heuristic Offloading Method called HOM was developed to assign deep learning tasks to a proper computing infrastructure in 5G networks [17]. The key factor of offloading decision is the transmission delay that is changed by the path length between a mobile device and a deep learning computational unit. During the offloading path identification, the transmission delay of each path is estimated in a heuristic manner. The path is confirmed by using the shortest offloading path finding mechanism.

Li et al. proposed an edge computing-based deep learning framework for the Internet of things [18]. Edge servers conduct several learning network layers to reduce input size until the input size is enough to be small to send data to the centralized cloud. Scheduling between IoT devices and an edge server is considered the service capacity and network bandwidth. If the service capacity of an edge server and the network bandwidth between an IoT device and an edge server are enough to handle an n-layered

network task, the whole task is offloaded to the edge server. In the case of not enough to handle an n-layered network task, it checks the $(n-1)$-layered network repeatedly.

DeepWear was proposed to decide on deep learning tasks for wearable devices [19]. A wearable device sends its task to a paired mobile device because wearable devices do not have a direct connection to the Internet. Under these environments, DeepWear supports partial offloading, context-aware scheduling, and pipelined processing to utilize the resources of the paired mobile device.

3. Proposed Computation Offloading

3.1. Our System Models and Assumptions. In our proposed system model, an AI-powered drone uses a mounted camera to monitor its remote areas, such as rivers. Figure 1 shows an overview of our system. Drones monitor their target field and analyze image data. Analyzing image data with a deep learning mechanism requires a different number of resources for each layer. One layer can handle on-device resources, but another layer cannot handle within the deadline. Our system, that is, applied offloading mechanism, can optimize resource consumption by deciding whether to offload each layer or not.

The AI-powered drone can analyze the problems in the monitoring area using the captured picture and the CNN (Convolution Neural Network) for analysis. The AI-powered drone can transmit the result to the mobile base station. An AI-powered drone can process inferences using CNNs locally or offload some of the inference jobs to servers with GPUs that are more computationally powerful than itself. For example, if the CNN-based inference model is composed of N layers, an inference job can be processed to the n-th layer on the drone. The GPU server can perform the rest of the inference job from the $n+1$-th layer to the N-th layer, where $O \le n \le N$. It is also assumed that multiple AI-powered drones are operating simultaneously. We assume that each inference task is given a deadline to complete the task.

3.2. Expected Execution Time Model. The expected execution time of an inference task by offloading can be calculated according to our model as follows: $\mathrm{ET}_{\mathrm{total}}(n)$ denotes the expected execution time of an inference job when the AI-powered drone processes the inference job until the n-th layer and offloads the rest of the inference job to a GPU server where $0 \le n \le N$. $S(n)$ means the total amount of computing operation to process up to the n-th layer. C_{drone} denotes the computing power in terms of the clock speed of the AI-powered drone. $D(n)$ denotes the total size of the intermediate result data inferred up to the n-th layer. B_{drone} denotes the network bandwidth between the AI-powered drone and the GPU server. $T_{\mathrm{server}}(n)$ means the time required to perform the rest of the inference task after the n-th layer on the GPU server. Since we can assume that we know the GPU server's specifications, $T_{\mathrm{server}}(n)$ can use a precomputed value. $\mathrm{ED}_{\mathrm{server}}$ denotes the expected processing delay of the GPU server like queuing delay due to multiple inference requests. $\mathrm{ED}_{\mathrm{server}}$ value can be estimated by sending an inference request from each drone and measuring the time to return the inference result. $\mathrm{ED}_{\mathrm{server}}$ excludes $T_{\mathrm{server}}(n)$, which can be calculated and used in advance. The cost for receiving the recognition result is a small constant value and is not considered in the expected execution time model:

$$\mathrm{ET}_{\mathrm{total}}(n) = \frac{S(n)}{C_{\mathrm{drone}}} + \frac{D(n)}{B_{\mathrm{drone}}} + T_{\mathrm{server}}(n) + \mathrm{ED}_{\mathrm{server}}. \quad (1)$$

3.3. Expected Energy Consumption Model. Since AI-powered drones are battery-operated, energy consumption must be considered when offloading inference tasks. The expected energy consumption of an inference task by offloading can be calculated as follows: N denotes the total number of layers of a given inference model. $\mathrm{EEC}_{\mathrm{total}}(n)$ denotes the expected energy consumption of an inference job when the AI-powered drone processes the inference job until the n-th layer and offloads the rest of the inference job to a GPU server where $0 \le n \le N$. $P_{\mathrm{processing}}$ denotes the power consumption (unit is Watt) per unit time of an AI-powered drone. P_{transmit} denotes the power consumption for transmitting data in an AI-powered drone. Therefore, the total expected energy consumption of the AI-powered drone can be calculated, as shown in the following equation:

$$\mathrm{EEC}_{\mathrm{total}}(n) = \frac{S(n)}{C_{\mathrm{drone}}} \times P_{\mathrm{processing}} + D(n) \times P_{\mathrm{transmit}}. \quad (2)$$

3.4. Proposed Inference Offloading Scheme. Algorithm 1 shows our inference offloading scheme for AI-powered drones. We can find the layer number n that minimizes the energy consumption of the AI-powered drone while meeting a given deadline constraint. $\mathrm{EEC}_{\mathrm{max}}$ denotes the required maximum energy consumption for an inference job offloading. $\mathrm{EEC}_{\mathrm{min}}$ indicates the required minimum energy consumption for an inference job in the AI-powered drone. DT denotes a deadline constraint of an inference job. $\mathrm{Offloading}_{\mathrm{point}}$ means the layer number of the inference model which minimizes the energy consumption of the AI-powered drone while meeting the deadline.

4. Evaluation

4.1. Simulation Environments. For the evaluation of the proposed scheme, we measured the execution time and energy consumption of a deep learning model in Raspberry Pi 3 with Google Coral, which is a popular NPU (Neural Processing Unit) for embedded devices. We used a model based on VGG16 as a deep learning algorithm and measured energy consumption using the Monsoon power monitor, which is widely used as a measurement of energy for mobile devices. The information and measured values of each layer in the deep learning model can be found in Table 1.

We assumed that the drone operates in the form of performing recognition by photographing the area to be sensed in the size of $384 \times 384 \times 3$ (width × height ×

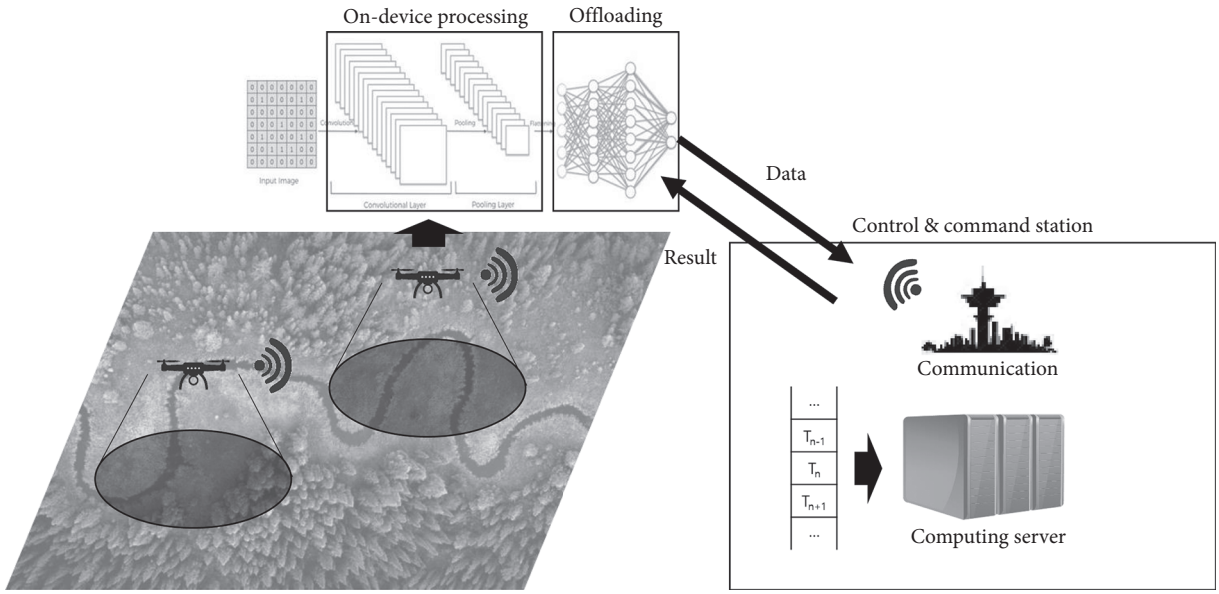


Figure 1: System overview.

(i) Result: $\text{Offloading}_{\text{point}}$//layer number n to offload
(ii) $\text{EEC}_{\text{min}} = \text{EEC}_{\text{max}}$
(iii) $\text{Offloading}_{\text{point}} = 0$
(iv) $n = 0$
(v) while $n \leq N$ do
(vi) Calculate $\text{ET}_{\text{total}}(n)$
(vii) Calculate $\text{EEC}_{\text{total}}(n)$
(viii) if $\text{EEC}_{\text{total}}(n) \leq \text{EEC}_{\text{min}}$ then
(ix) if $\text{ET}_{\text{total}}(n) \leq \text{DT}$ then
(x) $\text{Offloading}_{\text{point}} = n$
(xi) $\text{EEC}_{\text{min}} = \text{EEC}_{\text{total}}(n)$
(xii) end
(xiii) end
(xiv) $n = n + 1$
(xv) end.

Algorithm 1: Find n while meeting deadline constraints and minimizing energy consumption in an AI-powered drone.

Table 1: Information and measured values of each layer in the deep learning model.

Layer #	Layer type	Dimension and size of output	Execution time (s)	Power (J)
Input	—	(384, 384, 3)	0.1267	—
1	C	(112, 112, 64)	0.1346	0.546
2	C	(112, 112, 64)	0.0107	0.256
3	M	(56, 56, 64)	0.0801	0.265
4	C	(56, 56, 128)	0.1727	0.471
5	C	(56, 56, 128)	0.0120	0.540
6	M	(28, 28, 128)	0.0150	0.005
7	C	(28, 28, 256)	0.1336	0.209
8	C	(28, 28, 256)	0.2562	0.626
9	C	(28, 28, 256)	0.0352	0.881
10	M	(14, 14, 256)	0.0767	0.006
11	C	(14, 14, 512)	0.1413	0.366
12	C	(14, 14, 512)	0.1826	0.604
13	C	(14, 14, 512)	0.0087	0.731
14	M	(7, 7, 512)	0.3299	0.034
15	F	(4096)	0.0481	1.198
16	F	(2048)	0.0374	0.203
17	F	(2)	0.1346	0.147

*Layer type: C (convolution), M (max pooling), and F (fully connected).

channels) in succession using the camera mounted on the drone. The data taken by the drone while flying for about 10 minutes are about 8 GB.

We assumed that the server's processing power is 10 times or 1000 times better than the embedded device equipped with a drone. We also assumed that the queuing delay in the server is 500 ms on average and 1000 ms on maximum. The average queuing delay of the GPU server is set to an average of 500 ms, which is based on the server queuing model proposed by Lu et al. [20]. The delay is caused by the server's processing of multiple requests from drones simultaneously.

We assumed the network between the drone and server is Wi-Fi and measured the transmission time and the transmission energy in Raspberry Pi 3. We used the measured values as simulation parameters. The transmission energy used in the simulation was 3.335 J/MB, and the transmission rate was 10 MB/s.

In the experiment, we conducted a simulation of four schemes: Local-only, Server-only, Offloading, and Realtime-Offloading, and compared results. A total of 10,000 simulations were performed, and the results were averaged. In the case of the Local-only scheme, the drone does not offload by completing all executions on the drone. In the case of the Server-only scheme, the drone transfers images of the camera to the server without any execution of the deep learning model. In the case of the Offloading scheme, the drone runs the existing computation offloading scheme that only considers minimizing energy consumption without considering real-time constraints. In the case of the Offloading scheme, the offloading point is always fixed because it aims to minimize energy consumption. In the given experimental environment, the Offloading scheme processes up to the 6-*th* layer of the CNN model and offloads subsequent computations to the GPU server. In the case of the Realtime-Offloading scheme, the proposed scheme, the drone finds an optimal layer where the real-time constraints are to be met while considering minimizing energy consumption. Therefore, the proposed scheme's offloading point varies according to real-time constraints.

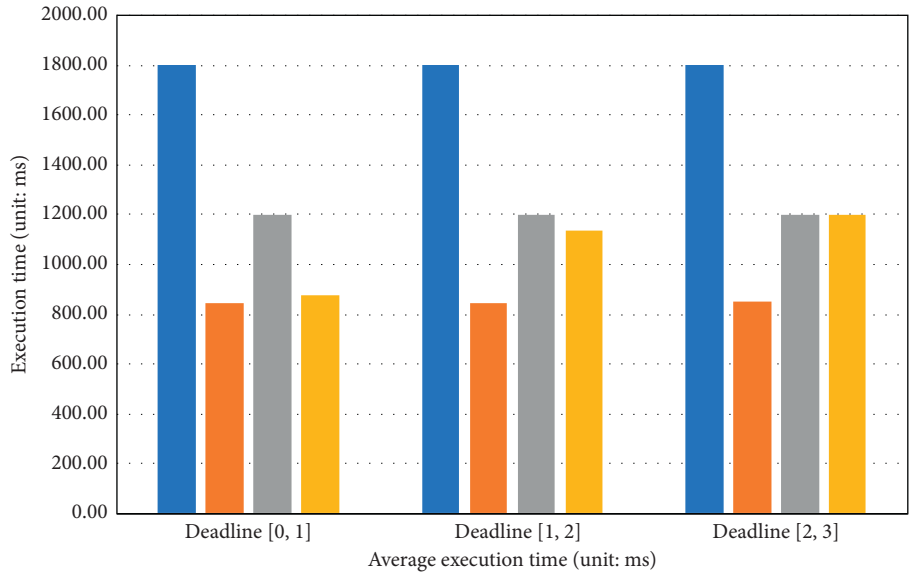
4.2. Simulation Results. In this section, we evaluate the performance of the Realtime-Offloading scheme through the simulation based on experimental energy consumption and execution studies. Our detailed simulation study showed that the Realtime-Offloading scheme meets the deadline considering energy consumption better than the existing schemes. We compare the Realtime-Offloading scheme, Local-only, Server-only, and existing offloading across a range of deadlines. In our experiment, we simulate a situation in which the deadline times are generated from a random distribution on $[m, m+1]$. As the m value increases, the deadline time required for an inference job increases. We used the deadline meet ratio as a real-time metric. To quantify the real-time characteristic of offloading schemes, the simulation measured the ratio between the number of jobs executed to meet the deadline constraint and the total number of inference jobs during some time interval.

Figure 2 shows the comparison of the execution time according to the deadline randomly assigned in the range of [0, 1], [1, 2], and [2, 3] for existing schemes. The range of execution times varies with the deadline times for all algorithms. The Local-only scheme has a very large execution time among the existing ones. This means that inference jobs with relatively large running times are not suitable to perform only on a drone. While the Server-only scheme appears to have the smallest average execution time, the energy consumption can be very high as all input data must be transferred to the server. Our Realtime-Offloading scheme has a similar or small execution time on average compared to the existing offloading scheme for various deadline ranges. This means that our scheme can be suitable for applications that meet the requirements very well in time.

Figure 3 shows the average energy consumption according to the deadline requirements for several schemes. Among other schemes, the Local-only scheme shows the largest average energy consumption. The energy consumption for Local-only is 7.09 J/operation. The result remains high regardless of whether the deadline time increases. That is why offloading is needed in battery-based drones. In particular, the Server-only scheme also requires the transfer of large amounts of input data regardless of the deadline, which consumes a lot of transmission energy. It can also be an unacceptable overhead considering the communication environment between drones and the offloading server where the network may be unstable. The Realtime-Offloading scheme offloads the server while considering the deadline, resulting in relatively slightly larger energy consumption compared to the existing offloading schemes. However, as the deadline increases in time, energy consumption can be decreased.

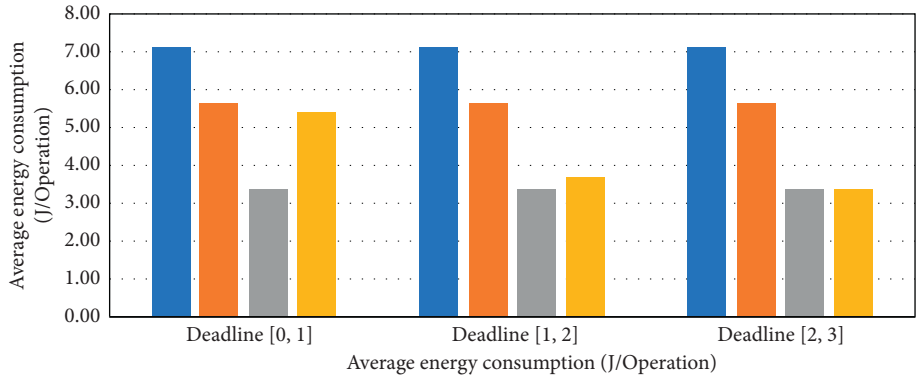
Figure 4 shows that Realtime-Offloading scheme gives the best performance among the evaluated in terms of the deadline meet ratio and energy consumption. This figure shows the deadline meet ratio according to the deadline requirements for Local-only, Server-only, existing offloading, and Realtime-Offloading scheme. For Server-only, the transmission consumes a lot of energy, but we assume that the performance of the server is very high, so real-time constraints can mostly be satisfied for cases where the deadline >1. Our experiments show that, for these cases, the proposed technique can obtain high deadline meet ratios with 65% at most of the energy consumption of Server-only. Compared with the existing offloading scheme, we confirm that the energy consumption of the proposed scheme is slightly larger, but the real-time constraints are well-matched regardless of all deadline requirements. Furthermore, as the timely requirements of the deadline increase, energy consumption decreases because optimal layers can be found to minimize energy consumption. Note that our Realtime-Offloading finds the layer number n that minimizes the energy consumption of the AI-powered drone while meeting a given deadline constraint.

Figure 5 shows the deadline meet ratio according to the server's processing power. In this experiment, the deadline is randomly generated from the range of [1, 3], assuming that the performance of the server is 10, 100, and 1000 times the



	Deadline [0, 1]	Deadline [1, 2]	Deadline [2, 3]
Local-only	1801.63	1801.63	1801.63
Server-only	845.59	847.57	850.92
Offloading	1198.39	1200.38	1198.35
Realtime-offloading	873.72	1133.78	1198.35

Figure 2: The execution time according to the deadline requirements.

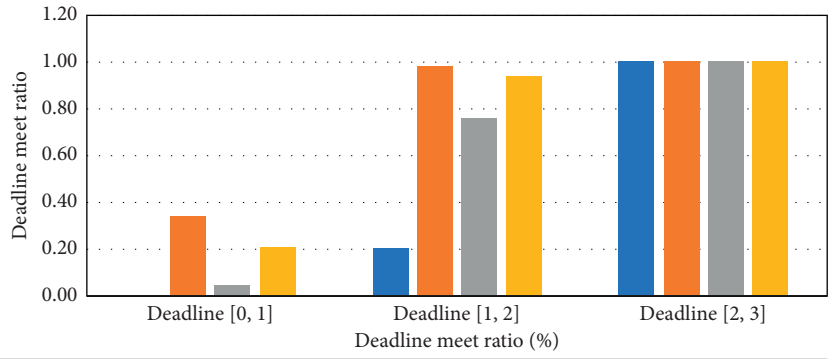


	Deadline [0, 1]	Deadline [1, 2]	Deadline [2, 3]
Local-only	7.09	7.09	7.09
Server-only	5.63	5.63	5.63
Offloading	3.36	3.36	3.36
Realtime-offloading	5.37	3.67	3.36

Figure 3: The average energy consumption for several schemes.

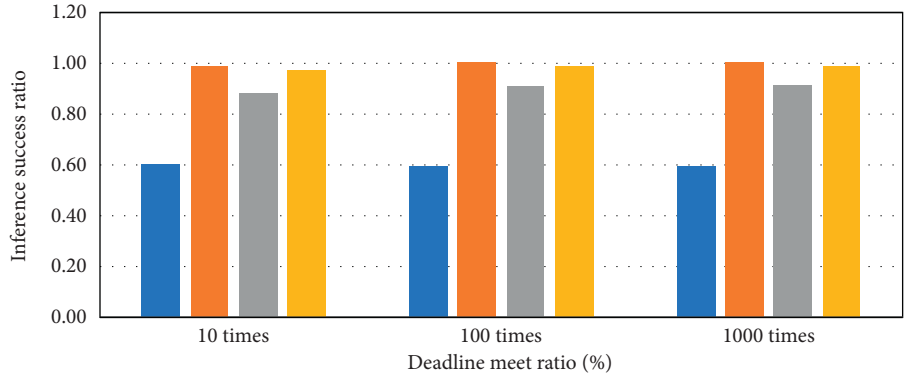
local performance, respectively. All schemes show that the deadline meet ratio increases as the server's processing power increases. This is because the average execution time is shortened due to the increased processing power of the server. More importantly, the rate of increase of Proposed-Offloading in the deadline meet ratio was the highest as the processing power of the server increased. The simulation result shows that the Proposed-Offloading has a deadline meet ratio of at least 97% with 61% of the energy consumption of Server-only scheme. This means that the Proposed-Offloading scheme is suitable for inference applications with deadline constraints.

Figure 6 shows the deadline meet ratio for the server's queuing delay of average of 100 ms, 500 ms, and 1000 ms, which are randomly distributed from (0, 200], (0, 1000], and (0, 2000]. The figure shows that, as the average queuing delay increases on the server, the deadline meet ratio of all schemes is reduced except for Local-only. Note that Local-only is not affected by queuing delays on the server because inference requests are handled locally. Even if the queuing delay increases, the decrease rate of Realtime-Offloading is relatively lower than that of existing offloading schemes, which satisfies the deadline requirements well.



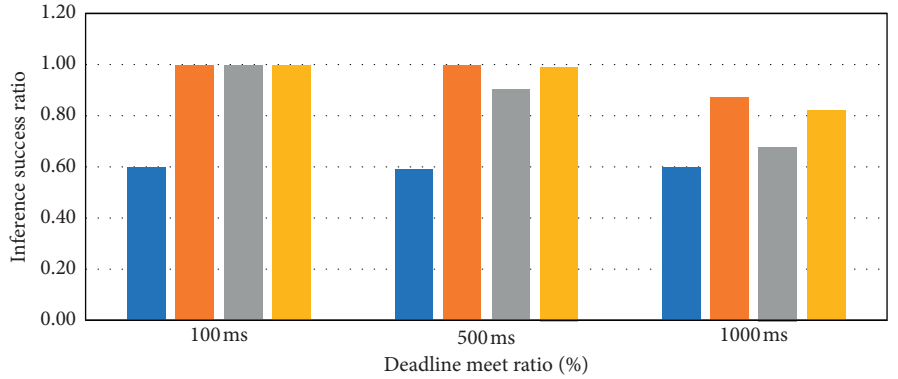
	Deadline [0, 1]	Deadline [1, 2]	Deadline [2, 3]
Local-only	0.00	0.20	1.00
Server-only	0.34	0.98	1.00
Offloading	0.05	0.76	1.00
Realtime-offloading	0.21	0.94	1.00

Figure 4: The deadline meet ratio according to the deadline requirements.



	10 times	100 times	1000 times
Local-only	0.60	0.59	0.59
Server-only	0.99	1.00	1.00
Offloading	0.88	0.91	0.91
Realtime-offloading	0.97	0.99	0.99

Figure 5: The deadline meet ratio according to server's processing power.



	100 ms	500 ms	1000 ms
Local-only	0.60	0.59	0.60
Server-only	1.00	1.00	0.87
Offloading	1.00	0.91	0.68
Realtime-offloading	1.00	0.99	0.82

Figure 6: The deadline meet ratio according to server's queuing delay.

5. Conclusions

In this paper, we proposed an energy-efficient and real-time remote sensing in AI-powered drones. The proposed scheme can offload a deep learning-based analysis job while minimizing the energy consumption of the AI-powered drone and meeting a given deadline constraint. It calculates the time to perform the deep learning-based analysis algorithms to the specific layer, the time to transmit intermediate result data to the server, and the time to wait on the server. Then, it calculates the energy consumed by performing the analysis algorithm on drones and the energy consumption required to transmit the intermediate result data to the server. Using these calculations, we find the optimal layer to minimize energy consumption while meeting the deadline. Experimental results showed that the proposed scheme satisfies the real-time constraints while reducing energy consumption. It also showed that the proposed scheme satisfies the real-time constraints even when multiple drones send requests to the server.

We plan to model the communication delay caused by multiple drones' connections to GPU servers in more detail and study more sophisticated computational offloading techniques in future works.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors thank DuHyeuk Chang who helped with the experiment. This work was supported by the Korea Agency for Infrastructure Technology Advancement (KAIA) grant funded by the Ministry of Land, Infrastructure and Transport (Grant 20DPIW-C153746-02) and Gachon University.

References

[1] H. Min, J. Jung, B. Kim, J. Hong, and J. Heo, "Dynamic rendezvous node estimation for reliable data collection of a drone as a mobile IoT gateway," *IEEE Access*, vol. 7, pp. 184285–184293, 2019.

[2] Y. Nimmagadda, K. Kumar, Y.-H. Lu, and C. G. Lee, "Real-time moving object recognition and tracking using computation offloading," in *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2449–2455, Taipei, Taiwan, October 2010.

[3] J. R. Kellner, J. Armston, M. Birrer et al., "New opportunities for forest remote sensing through ultra-high-density drone lidar," *Surveys in Geophysics*, vol. 40, no. 4, pp. 959–977, 2019.

[4] C. Moussa, H. Zhang, H. Shamim, and K. Liu, "Task offloading strategies based on workload balancing in ultra-dense networks," in *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, pp. 217–223, Association for Computing Machinery, New York, NY, USA, September 2019.

[5] H. Wu and K. Wolter, "Stochastic analysis of delayed mobile offloading in heterogeneous networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 2, pp. 461–474, 2018.

[6] B. Kim, H. Min, J. Heo, and J. Jung, "Dynamic computation offloading scheme for drone-based surveillance systems," *Sensors*, vol. 18, no. 9, 2018.

[7] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," *IEEE Wireless Communications Letters*, vol. 6, no. 6, pp. 774–777, 2017.

[8] H. Guo, J. Liu, J. Zhang, W. Sun, and N. Kato, "Mobile-edge computation offloading for ultradense iot networks," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4977–4988, 2018.

[9] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: a deep learning approach," in *Proceedings of the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–6, Montreal, QC, Canada, October 2017.

[10] X. Wang, X. Wei, and L. Wang, "A deep learning based energy-efficient computational offloading method in internet of vehicles," *China Communications*, vol. 16, no. 3, pp. 81–91, 2019.

[11] D. S. Rani and M. Pounambal, "Deep learning based dynamic task offloading in mobile cloudlet environments," *Evolutionary Intelligence*, vol. 165, 2019.

[12] X. Zhao, K. Yang, Q. Chen et al., "Deep learning based mobile data offloading in mobile edge computing systems," *Future Generation Computer Systems*, vol. 99, pp. 346–355, 2019.

[13] 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 of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.

[14] Z. Ning, P. Dong, X. Wang, J. J. P. C. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: an intelligent offloading system," *ACM Trans. Intell. Syst. Technol*, vol. 10, no. 6, 2019.

[15] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile networks and applications*, vol. 18, pp. 1–8, 2018.

[16] Y. Kang, J. Hauswald, C. Gao et al., "Neurosurgeon," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[17] X. Xu, D. Li, Z. Dai, S. Li, and X. Chen, "A heuristic offloading method for deep learning edge services in 5g networks," *IEEE Access*, vol. 7, pp. 734–744, 2019.

[18] H. Li, K. Ota, and M. Dong, "Learning iot in edge: deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[19] M. Xu, F. Qian, M. Zhu et al., "DeepWear: adaptive local offloading for on-wearable deep learning," *IEEE Transactions on Mobile Computing*, vol. 19, no. 2, pp. 314–330, 2020.

[20] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, and A. Greenberg, "Join-Idle-Queue: a novel load balancing algorithm for dynamically scalable web services," *Performance Evaluation*, vol. 68, no. 11, pp. 1056–1071, 2011.