electronics

MDPI

Article

Task Offloading of Deep Learning Services for Autonomous Driving in Mobile Edge Computing

Jihye Jang, Khikmatullo Tulkinbekov and Deok-Hwan Kim *

Department of Electrical and Computer Engineering, Inha University, Incheon 22212, Republic of Korea; jihye@inha.edu (J.J.); 22202364@inha.edu (K.T.)
* Correspondence: deokhwan@inha.ac.kr; Tel.: +82-32-860-7424

Abstract: As the utilization of complex and heavy applications increases in autonomous driving, research on using mobile edge computing and task offloading for autonomous driving is being actively conducted. Recently, researchers have been studying task offloading algorithms using artificial intelligence, such as reinforcement learning or partial offloading. However, these methods require a lot of training data and critical deadlines and are weakly adaptive to complex and dynamically changing environments. To overcome this weakness, in this paper, we propose a novel task offloading algorithm based on Lyapunov optimization to maintain the system stability and minimize task processing delay. First, a real-time monitoring system is built to utilize distributed computing resources in an autonomous driving environment efficiently. Second, the computational complexity and memory access rate are analyzed to reflect the characteristics of the deep learning applications to the task offloading algorithm. Third, Lyapunov and Lagrange optimization solves the trade-off issues between system stability and user requirements. The experimental results show that the system queue backlog remains stable, and the tasks are completed within an average of 0.4231 s, 0.7095 s, and 0.9017 s for object detection, driver profiling, and image recognition, respectively. Therefore, we ensure that the proposed task offloading algorithm enables the deep learning application to be processed within the deadline and keeps the system stable.

Keywords: task offloading; task allocation and scheduling; deep learning services; autonomous vehicle; Lyapunov optimization; Internet of Things (IoT)

Citation: Jang, J.; Tulkinbekov, K.; Kim, D.-H. Task Offloading of Deep Learning Services for Autonomous Driving in Mobile Edge Computing. *Electronics* **2023**, *12*, 3223. https://doi.org/10.3390/electronics12153223

Academic Editors: Jihoon Yang and Unsang Park

Received: 29 June 2023
Revised: 19 July 2023
Accepted: 25 July 2023
Published: 26 July 2023

1. Introduction

With the rapid development of artificial intelligence and autonomous driving technology today, autonomous vehicles do not simply provide driving and control functions. It is necessary to provide more complex and heavy tasks like applications that require deep learning inference, such as identifying drivers or recognizing objects on the road [1].

High computational task processing requires abundant computing resources and energy, but autonomous vehicles have limitations because the main energy source is a battery. In addition, autonomous vehicles must minimize vehicle volume and weight to maximize energy efficiency, so computing resources are naturally limited. Therefore, to provide deep learning services in an autonomous driving environment, a server with more energy and computing resources is required [2].

System designers can adopt a distributed computing architecture to handle high-computation tasks in constrained mobile devices such as autonomous vehicles. In cloud computing, a representative example of distributed computing, a cloud server with abundant computing resources supplements the computational power of mobile devices to enable load management and fast computational processing. However, the physical distance between the mobile user and the cloud server is very far, resulting in huge data transmission delays and network costs, so it is unsuitable for an autonomous driving environment. On the other hand, in mobile edge computing, edge clusters are located between mobile users and cloud servers to compensate for the disadvantages of cloud computing.

An edge cluster refers to a computing server consisting of small devices with limited computing resources but richer than mobile devices. Edge clusters are located close to mobile devices, reduce network latency and distribute the load by primarily processing tasks and data. Therefore, mobile edge computing is being actively studied as a promising solution for an autonomous driving environment [3].

When the edge cluster and cloud server process tasks instead of the mobile user in mobile edge computing, it is called task offloading. The system can improve the service experience of mobile users through task offloading by running deep learning applications even with limitations of space, computing resources, and energy [4]. Task offloading brings both profits and costs because server conditions such as computing resource limitation, data transfer delay, energy consumption, and network cost differ depending on which computing server is selected as the target server. Therefore, when designing a task offloading algorithm, it should be designed in consideration of the trade-off relationship between profits and costs and user requirements.

In particular, deep learning services used in autonomous driving are closely related to safety, so they are sensitive to system stability and delay [5]. In addition, since the neural network model and the transmission data are different for each service, the type and size of computing resources required for task processing are also different. For example, a driver profiling application typically used in autonomous driving uses time-series data obtained from internal and external sensors of the vehicle, so it has lower network costs. As another example, object detection application is characterized by the need to detect objects in real-time by vehicle mobility and requires complex computing operations for image data processing. Therefore, when task offloading of deep learning service is performed in mobile edge computing for an autonomous driving environment, the offloading algorithm must be designed considering the tasks' characteristics and the servers' status.

In previous studies, task offloading policies were studied from various perspectives. Among the studies that subdivided and formulated offloading procedures to offload complex applications, Chen et al. proposed an offloading scheme that utilizes a software-defined network in ultra-dense networks [6]. This research saves 30% of battery life and reduces latency by 20% in mobile devices compared to random and uniform task offloading schemes. Alameddine et al. improve the runtime of instances by dividing the offloading procedure into task allocation and execution order scheduling, considering the requirements of offloaded tasks and limited functions of mobile edge computing [7].

Research on task offloading that reflects the characteristics of the autonomous driving environment is also being actively conducted. Li formulated a task offloading problem in a WiFi-cellular heterogeneous network environment and optimized the trade-off relationship between offloading profits and costs [8]. Liu proposed a task scheduling algorithm that considers the vehicle's mobility in an autonomous driving environment and the limited computing resources of a mobile edge computing server to ensure that tasks were completed within a specified time [9]. However, these studies lack discussion on the deep learning applications that are offloaded and real-time characteristics in autonomous driving environments. Recently, researchers have been studying the offloading algorithm using reinforcement learning. Reinforcement learning has the advantage that it can be used without environmental information. But it requires a lot of training data and runtime. This paper proposes a task offloading algorithm to keep the system stable and minimize delay by monitoring the system status and analyzing the characteristics of deep learning services used in autonomous driving to overcome this weakness.

Figure 1 shows the task offloading system of the deep learning services for autonomous driving proposed in this paper. For task offloading of complex applications, the offloading steps are subdivided into 1. real-time monitoring, 2. task analysis, and 3. optimal decision-making. The offloading algorithm determines the best target server for processing the requested task from the system stability and delay side.

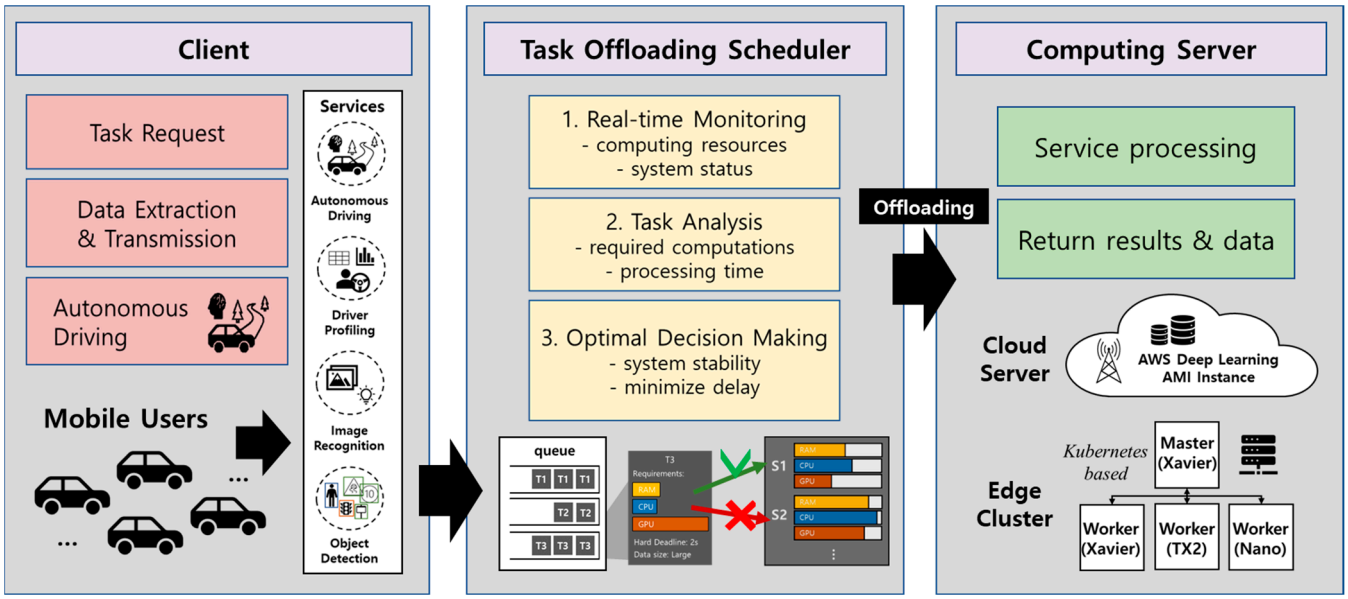

Figure 1. The architecture of the proposed task offloading system for autonomous driving.

The contributions of this work are as follows:

- Considering the characteristics of deep learning applications for autonomous driving, an offloading utility function is defined.
- A mobile edge computing architecture for the autonomous driving service is designed. It consists of a driving simulator(mobile user), Kubernetes-based edge cluster(edge server) and the remote server(cloud server).
- A Lyapunov optimization based task offloading algorithm is proposed to optimize the trade-off relationship between the system stability and the offloading utility. The performance of task offloading is guaranteed by means of mathematically proving the Lyapunov optimization framework.
- The developed mobile edge computing architecture can be applied to real network environment such as NVIDIA Jetson series embedded boards and an AWS cloud instance.

The remains of this paper are organized as follows. In Section 2, the trends in task offloading research are described. Section 3 analyzes each deep learning services used in autonomous driving to consider the characteristics to the task offloading algorithm. Section 4 describes the proposed mobile edge computing architecture and defines the system model and the offloading utility function using queue. Section 5 utilizes Lyapunov optimization to optimize the trade-off relationship between system stability and user requirements. Section 6 verifies the performance of the proposed algorithm through experiments. Finally, in Section 7, conclusions and future research are described.

2. Related Work

In an autonomous environment, task offloading is introduced to complement the computing resource and space constraints of autonomous vehicles. Task offloading refers to transferring an application or part thereof to a remote server with abundant computing resources for processing and then returning a result to overcome the limitations of the computing resources and calculation speed of the local device.

Although task offloading facilitates resource and load management for mobile users, profits and costs occur simultaneously in the system [10]. For example, when a mobile user offloads a task to a cloud server, the cloud server has abundant computing resources and processes high-computation tasks very quickly. However, the data transmission delay is significant due to a long physical distance from the mobile user. Conversely, if the task is offloaded to the edge cluster, the edge cluster is physically close to the mobile user, so

there is less data transmission delay. But the edge cluster has fewer computing resources, so the work processing speed is slow. Therefore, system designers should consider the data transfer delay, energy consumption, and network cost during task offloading and use a task offloading algorithm optimized for the user's requirements and system purpose.

Since mobile edge computing is distributed computing, there are various environmental variables such as machine status, computing resource availability, and random variables. If you observe and calculate all the environment variables, you can design a perfect task offloading. The demonstration of the task offloading in between autonomous vehicle and edge-computing infrastructure can be the good example of such varied variables [11]. However, it is impossible to know all the environmental variables that change from moment to moment. Also, user requirements considered in task offloading have a trade-off relationship, and there are various ways to optimize them. Many existing studies have designed an offloading strategy based on the Markov decision process. It makes an offloading decision by observing all probabilistic information of the system to optimize the trade-off relationship and achieve an objective function [12,13]. The Markov decision process performs the objective function by calculating and predicting the probability distribution of the system and generally makes efficient offloading decisions. Still, it is tough to apply to stochastic environments such as actual mobile edge computing environments.

Recently, research is being conducted to make offloading decisions using artificial intelligence [14]. There are methods for supervised learning by collecting vast amounts of system data, but most offloading policies must be able to be deployed without knowing system environment data. Therefore, research on offloading strategies using reinforcement learning, which learns based on subjects, states, actions, and rewards defined in an environment rather than learning according to learning data, is being actively conducted [15]. However, reinforcement learning requires countless tries, data, and time and needs much more in a complex stochastic environment. In addition, finding an analytic solution for system optimization for reinforcement learning is complicated and difficult.

On the other hand, Lyapunov optimization-based task offloading can make an offloading decision with only a small amount of observational information, such as queue backlog, arrival data size, and server computing resources capability, even without knowing the probability distribution of the system. Its simple algorithm makes it easy to deploy in various environments. Moreover, it is possible to mathematically prove and formulate the optimization and the task offloading step to ensure performance [16]. Therefore, we adopt the Lyapunov optimization technique to optimize the trade-off relationship between profit and cost during task offloading.

3. Task Analysis for Deep Learning Applications

In this section, the neural network model structures and characteristics of deep learning applications for autonomous driving are analyzed to reflect on the task offloading algorithm. We select three deep learning applications mainly used in autonomous driving services and containerize and distribute them so they can be processed on edge clusters and cloud servers. Driver profiling is one of the widely used applications that improves the overall quality of autonomous driving. For this purpose, the time-series data from driver's response in different situations like changing lanes, taking a break or acceleration, taking turns or emergencies are collected and used to improve the quality of autonomous driving. Additionally, Image recognition is another popular application used in autonomous driving. Using a proper implementation of image recognition, autonomous driving vehicles achieve to have features like following lates, auto-parking, detecting traffic lights, object tracking and overall scene understanding. The third popular application considered in this research is object detection. Even though it seems object detection can be covered by image recognition, the applied usage is totally different. While image recognition aims to recognizing and categorizing the images on the environment, object detection focuses on locating the specific object within the image and act. For example, using image recognition, the lanes can be seen, and overall route of the autonomous vehicle can be planned. And

using object detection application, the autonomous vehicles achieve to take actions like changing lanes, navigation based on traffic situation and collision avoidance.

In a mobile edge computing environment, model size, computational complexity, and inference time must be considered when designing a neural network model to provide deep learning services on mobile devices or embedded boards. Each deep learning service has different input data, as shown in Table 1, and the structure of the neural network model, so there are differences in required computing resources, inference time, and data communication latency. So, it is necessary to analyze the characteristics of deep learning applications and use them for task offloading algorithms.

Table 1. Input Data Type and Shape of Each Deep Learning Model.

	Driver Profiling	**Image Recognition**	**Object Detection**
Data type	time-series data	image data	image data
Data shape	(40, 15)	(224, 224, 3)	(416, 416, 3)

According to ShuffleNet V2 [17], it is pointed out that most neural network model performance evaluations rely on indirect metrics such as computational complexity like FLOPs (Floating point operations), but performance such as inference speed is also related to memory access and platform characteristics. Therefore, this research utilizes computational complexity and memory access rate to analyze the characteristics of each application and uses lightweight deep learning applications widely used in autonomous driving. The deep learning applications used in this paper are as follows:

- Driver profiling: It is a deep learning application that identifies the driver using scalar data generated inside and outside the vehicle while driving. We use a lightweight model using depthwise and sparse learning [18].
- Image recognition: It is a deep learning application that recognizes images of the driving view of a vehicle. We use Mobilenet V2, a lightweight version of MobileNet by Google [19].
- Object detection: It is a deep learning application that recognizes objects such as people, vehicles, and traffic lights and their positions by using the image of the driving view of the vehicle. Based on the YOLOv3 model, a model customized and retrained for the autonomous driving simulator environment is used [20].

3.1. Computational Complexity

In general, driver behavior recognition requires fewer computing resources because it does not require image processing in deep learning inference. On the other hand, image recognition and object detection use more computing resources because they use image data. In particular, object detection uses a lot of computing resources because it divides one image into numerous regions in the inference process, finds objects in each area, and performs localization. Table 2 shows these analysis results by measuring the parameter and FLOPs. The parameter means the number of parameters counted, and FLOPs implies the number of floating-point operations performed when processing a neural network model. These are generally proportional and are used as numerical values representing the computational complexity of neural network models.

Table 2. The Computation Complexity of Each Deep Learning Inference.

	Driver Profiling	**Image Recognition**	**Object Detection**
Parameters	119,832	3,538,984	65,252,682
FLOPs	0.233 MFLOPs	0.615 GFLOPs	6586 GFLOPs

3.2. Memory Access Rate

To analyze the memory access rate of each deep learning service, we first examine the lightweight process of each neural network model. Mobilenet V2, used for image

recognition, reduces the memory space required during inference by not fully materializing the intermediate tensor in the convolution module so that it can be utilized in embedded hardware [18]. In other words, it is a lightweight model that reduces the frequency of memory access in the previous model, Mobilenet V1. On the other hand, YOLOv3, used for object detection, is a model developed with the purpose of real-time, and the model used for driver profiling aims to reduce the amount of computation and the model size [19]. Therefore, it is judged that the memory access rate of image recognition will be small according to the lightweight purpose.

3.3. Measurement of Computing Resource Utilization

Since embedded devices constituting an edge cluster have different computing resource limitations, their processing capabilities are also different. To compare and analyze the characteristics of the three deep learning applications analyzed above, we measure the computing resource limitations of each server and the required computing resources for each task processing. Table 3 shows the computing resource limitations of each server. Table 4 shows the required computing resources to process each task in Jetson AGX Xavier.

Table 3. Computing Resource Limitations of the Edge Cluster.

	Xavier	TX2	Nano
RAM (MB)	39,927	7860	3964
CPU (MH/1 core)	1190 (8 cores)	345 (6 cores)	102 (4 cores)
GPU (MHz)	318	114	76

Table 4. Required Computing Resources for Each Deep Learning Inference.

	Driver Profiling	Image Recognition	Object Detection
RAM (MB)	3012.34	4672.84	1782.56
CPU (MH/1 core)	123	66.98	46.05
GPU (MHz)	13.10	10.48	163.44

Most of the computing resource utilization is similar to the analysis results, but one difference is that driver profiling uses more CPU and GPU than image recognition. It is expected as a result of the design purpose of the neural network model. In [17], the lightweight purpose of the driver profiling model is to reduce the size. On the other hand, image recognition lightweight considers the model size and the utilization of computing resources. Also, it is developed to be optimized for Keras API so that computing resource utilization can be minimized. Object detection uses much more GPU and less CPU, depending on the inference process.

Deep learning services use more computing resources than general tasks, and the required computing resources for each application are different. In addition, since mobile edge computing consists of servers with various computing resource limitations, it is important to assign appropriate servers to each task when offloading tasks. Therefore, the computational complexity and memory access rate discussed above are transmitted to the task offloading scheduler along with real-time and deadline constraints and are used for offloading decisions.

4. System Model

In this section, we design a mobile edge computing that enables autonomous vehicles to provide various deep learning applications to mobile users. The mobile edge computing is modeled as a queuing system. Also, we define the offloading utility function reflecting the characteristics of the tasks and various user requirements.

4.1. Design of the Mobile Edge Computing

Figure 2 shows an overview of mobile edge computing for the autonomous driving environment designed in this paper. Mobile edge computing consists of mobile devices, edge clusters, and cloud servers. As shown in the figure, the edge cluster is constructed by four Jetson embedded boards. For a better performance in running machine learning tasks, AGX Xavier, TX2, and Nano boards are selected for their GPU and computing capabilities. The edge cluster is constructed by a master (AGX Xavier) and workers. In the sample environment, we have added one instance of each board as the worker. Master node divides the task according to each board's capabilities and computing power. In this way, the edge cluster gets to achieve to execute different size tasks in heterogeneous environment. Moreover, Amazon AWS instance is used for the cloud implementation. Autonomous vehicles that request deep learning services are mobile users. An edge cluster is a collection of embedded devices close to mobile users. A cloud server has abundant computing resources located far from mobile users. When a deep learning service is requested from an autonomous vehicle, the request is processed by the edge cluster or cloud server. When multiple autonomous vehicles request deep learning services simultaneously, the request is received in the form of a stream, so the queue concept is introduced in Section 3 to allocate and manage tasks efficiently.

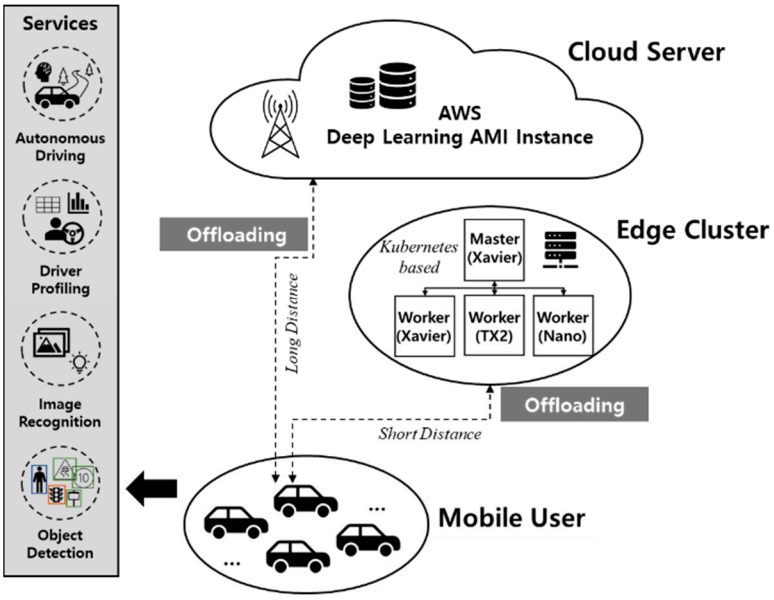

Figure 2. Mobile edge computing for autonomous driving environment.

To represent a mobile device, a game engine-based driving simulator CARLA (Car Learning to Act) [21] is used to build a virtual environment such as a city, vehicle, and people, as shown in Figure 3. It is designed to manage vehicle communication data between autonomous driving modules through ROS (Robot OS) [22] Bridge. The deep learning task request module is designed to transmit the requested service information to the external static IP of the task receiver. Data required for deep learning inference is transmitted through WebSocket bi-directional communication.

Deep learning applications on edge clusters are designed to be virtualized and executed in Linux containers to independently adjust various libraries and system environments for each application [23]. Therefore, the edge cluster was designed using a container orchestrator to facilitate the management of Linux container-type workloads and services. Among various container orchestrators, this paper adopts Kubernetes [24]. As an open-source platform, Kubernetes is highly deployable and scalable. In addition, it is used in various studies because it can automatically schedule the cluster distribution of components according to user requirements and automate configuration, management,

and failure handling [25–27]. A Kubernetes-based edge cluster consists of a master node that controls the entire cluster and worker nodes that receive commands from the master node and run actual services. A node is a worker machine in Kubernetes and can be a virtual or physical machine depending on the cluster.

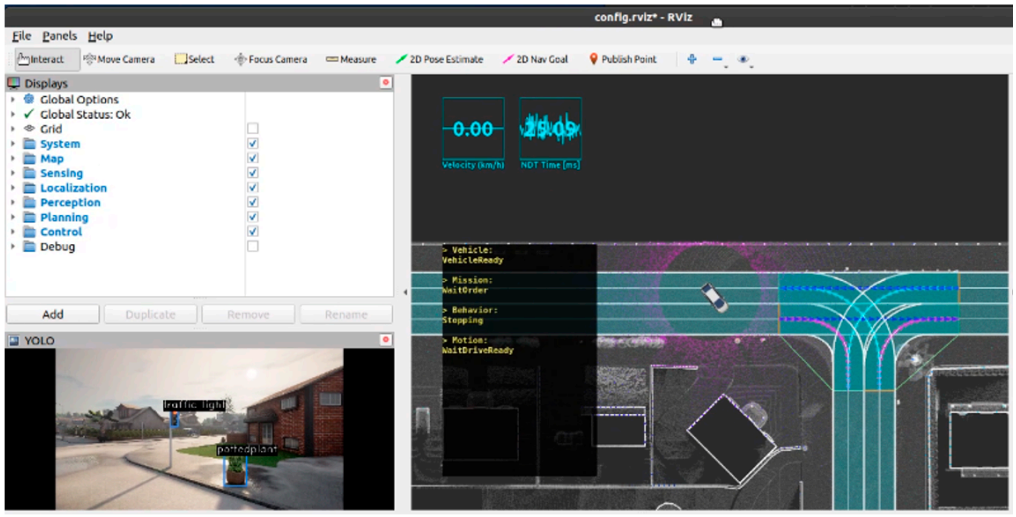

Figure 3. Autonomous driving environment using CARLA simulator and ROS.

Embedded boards are used to construct a physical machine in this research. When a mobile device requests a task to the edge cluster, the master node monitors the state of the edge cluster and selects an appropriate worker node for task processing according to the scheduler and the strategy set by the user. Tasks are composed of Docker [28] containers and deployed to worker nodes in pods, as shown in Figure 4.

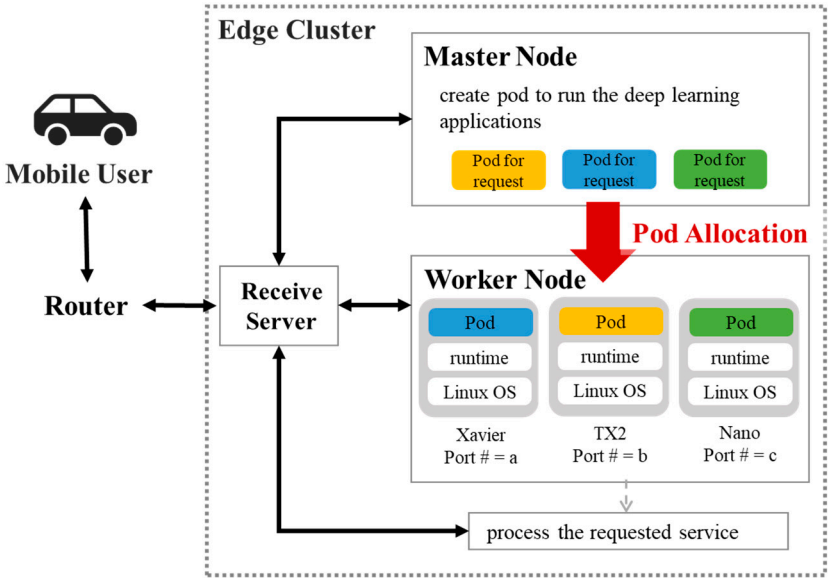

Figure 4. Pod allocation in edge cluster.

A pod is the smallest unit of an application that can be deployed in Kubernetes and is a collection of one or more Linux containers. A task is composed on several processes, and the Linux container is not capable of residing more than one process. So, the pod is employed as a smallest unit that packs several containers in higher level abstraction. A pod is created only when a server receives a task that has never been executed, and the pod continues to exist after that. After the pod is created, a WebSocket server is opened for

data communication. When the task is finished, the WebSocket connection is disconnected. Then, if a request for the same task comes in, the WebSocket is reconnected.

Kubernetes uses an internal virtual network to manage pod deployment and provide functions between applications. Figure 5 shows the internal network of the edge cluster designed in this paper. Through Service and Kube-proxy, the internal network supports pod allocation and connection according to each task. Service is a network service function that exposes an application running in Kubernetes as a fixed endpoint for external connectivity or supports load balancing between pods.

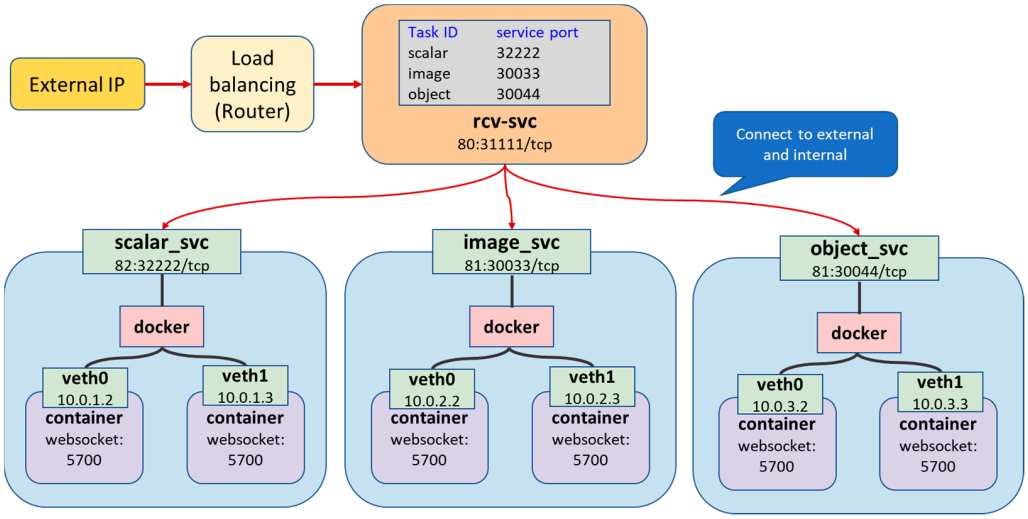

Figure 5. The internal network of edge cluster.

Since the cloud server has abundant computing resources and must be located away from mobile devices, it is built using a deep learning instance of AWS(Amazon Web Service). When a mobile device requests a task to the cloud server, the cloud server processes the task by creating a sub-process that loads the neural network model for the task.

4.2. Queueing Model

Figure 6 shows mobile edge computing as a queuing system. The notations are shown in Table 5. The system operates in discrete time and uses time slot Δt. Each component of the system is as follows:

- Arrival: The amount of deep learning service workload requested by an autonomous vehicle at time t. It is defined as $a_i(t)$. i is the task number. The requested operations are independent and identically distributed with each other. The expected value of the arrival for each task is defined as $\mathbb{E}\{a_1(t)\} = \lambda_1, \mathbb{E}\{a_2(t)\} = \lambda_2, \mathbb{E}\{a_3(t)\} = \lambda_3$.
- Queue backlog: It refers to a load of unprocessed tasks at time t waiting to be offloaded to the target server after a task is requested. It is defined as $Q_i(t)$. There is a data queue for buffering the offloading data for each task. At $t = 0$, the queue backlog is assumed to be zero. $Q_i(0) = 0$
- Departure: It refers to the amount of task that is offloaded to the server and processed at time t. It is defined as $b_i(t)$. Departure is determined by the task offloading algorithm and the server situation.
- Server: A server that can process the requested tasks. It is defined as S_j. j is the server number. In this system, server means the embedded devices constitute the edge cluster and cloud servers.

$$Q_i(t+1) = max[Q_i(t) - b_i(t),\ 0] + a_i(t) \tag{1}$$

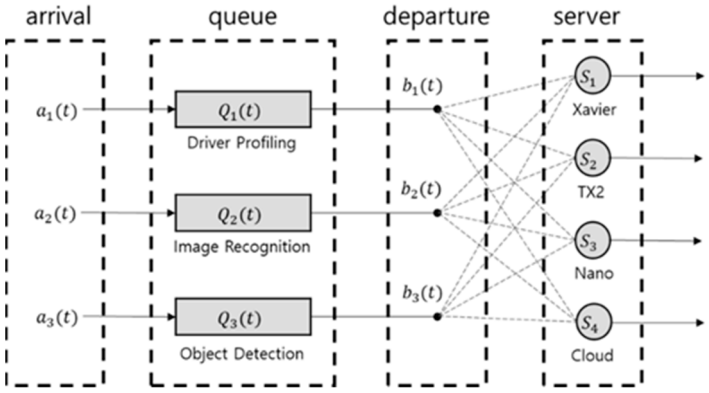

Figure 6. Queueing system model.

Table 5. The List of Main Notations.

Notation	Description
i	task number
M	the number of tasks
j	server number
$a_i(t)$	arrival data at time slot t
$b_i(t)$	departure data at time slot t
$Q_i(t)$	queue backlog at time slot t
S_j	server
λ_i	the value of time average of arrival
$U(t)$	offloading utility function
$P(t)$	offloading profit function
$C(t)$	offloading cost function
D_t	transmission delay
D_e	execution delay
α	weight parameter of task offloading profit
d_r	size of inference result data
β	parameter of task characteristics
BW	network bandwidth
R_{capa}	computing resource capability
v_i	Lagrange multiplier

Equation (1) is the queuing dynamics equation that predicts the amount of queue backlog at the next time slot $t+1$ by adding the number of arrivals entering the queue and subtracting the number of departures leaving the queue at the time slot t.

4.3. Offloading Utility Function

In this paper, task offloading is defined as transferring tasks between mobile devices and edge clusters or cloud servers by mapping mobile devices to local devices and edge clusters and cloud servers to remote servers.

Figure 7 shows the overview of the proposed task offloading. Autonomous vehicles, which are mobile devices, request multiple tasks simultaneously. Receiving server in the task offloading scheduler receives the stream-type task requests and allocates them to each offloading queue. Then the tasks wait until serviced. At this same time, the task offloading scheduler analyzes the tasks' characteristics and collects the edge servers' status using real-time monitoring. In conclusion, at the offloading decision-making step, the task processing deadline, computation complexity, memory access rate, and available computing resources of the computing server are considered to select the optimal task offloading target server using the Lyapunov optimization framework.

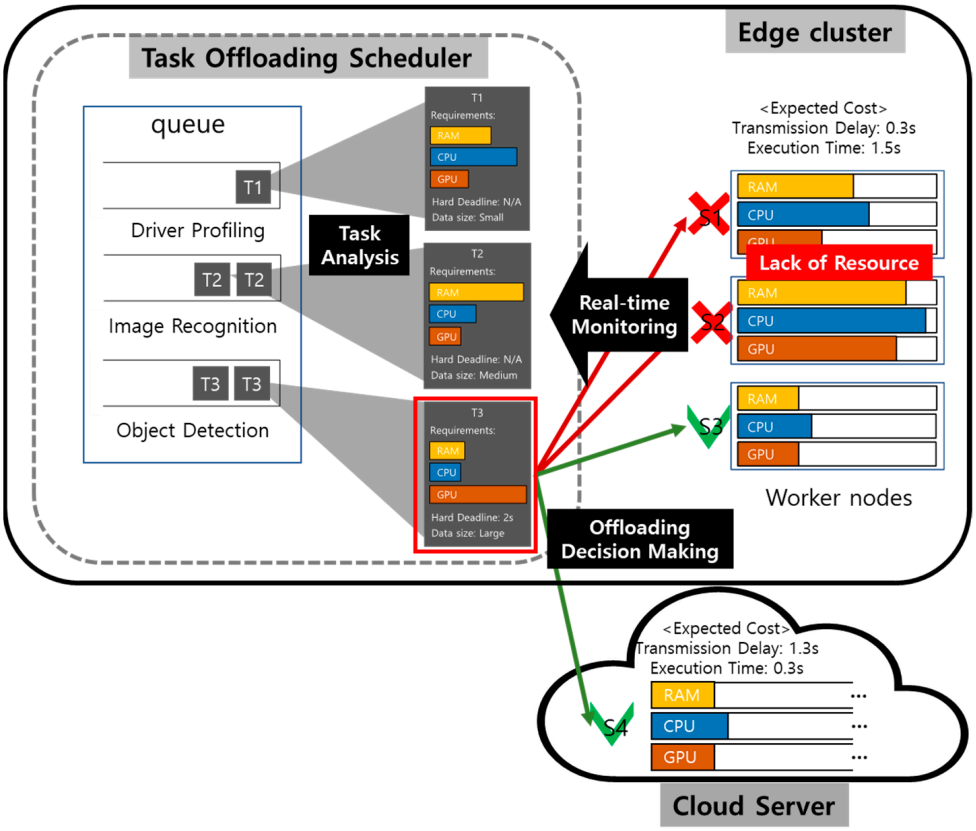

Figure 7. Overview of the proposed task offloading.

Task offloading has the profit of reducing the load on the server, but at the same time, costs such as latency and lack of computing resources. We define a profit function for offloading workloads and a cost function regarding latency and delay time.

The profit function is defined as $P(t)$, as shown in Equation (2). A diminishing returns function is used [29]. α is the weight parameter.

$$P(t) = \alpha log[1 + b_i(t)] \tag{2}$$

The delay caused by the task offloading can be classified into transmission delay and computing delay according to each characteristic and is defined as follows:

(1) Transmission delay: It means the network delay time by data transmission and is defined as (3). $b_i(t)$ means the size of offloading workload to be processed by being offloaded to the server, and d_r means the data size of the inference result. BW means network bandwidth.

$$D_t(t) = \frac{b_i(t) + d_r}{BW} \tag{3}$$

(2) Execution delay: It means the computation time by executing the deep learning service and is defined as in (4). β is a parameter reflecting the computational complexity and memory access rate of the deep learning service, and R_{capa} represents the computing resource capability of the computing server.

$$D_e(t) = \frac{b_i(t) \cdot \beta}{R_{capa}} \tag{4}$$

The total offloading cost is the sum of all delay functions caused and is defined as (5).

$$C(t) = D_t(t) + D_e(t) \tag{5}$$

Therefore, the task offloading utility function is defined as (6).

$$U(t) = P(t) - C(t) \tag{6}$$

5. Lyapunov Optimization Framework for Task Offloading

We propose a Lyapunov-based task offloading framework to solve a trade-off between system stability and the offloading utility with the delay constraints and computing resource limitations. In this section, we define the Lyapunov drift function and establish a trade-off relationship between system stabilization and the objective function with the Lyapunov optimization framework.

5.1. An Optimization Problem of Trade-Off Issues

This paper aims to design a task offloading algorithm that maximizes the utility of task offloading while keeping the system stable. System stabilization can be satisfied by maintaining the queue backlog, so it does not diverge. User requirements can be achieved by maximizing the task offloading utility function. Therefore, the final goal of task offloading in this research is shown in (7) and (8).

$$max \left(\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{U(t)\} \right) \tag{7}$$

$$s.t. \lim_{t\to\infty} sup \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\Big\{ \sum_{i=1}^{M} Q_i(\tau) \Big\} < \infty \tag{8}$$

To provide various deep learning services in an autonomous driving environment, task offloading aims to maximize the utility of task offloading with system stability. However, there is a trade-off between system stabilization and maximization of the task offloading utility. Therefore, this paper uses the Lyapunov optimization framework to optimize this trade-off relationship. Lyapunov optimization-based task offloading algorithms are easy to deploy in various environments because of their simple algorithms. Also, it has the advantage of systematically formulating the offloading step and guaranteeing performance because mathematical proof is possible [15].

Lyapunov optimization is an observational information-based framework, so it derives its algorithm based on the current queue backlog, i.e., the number of tasks to be processed. Therefore, the Lyapunov function $L(t)$ and the Lyapunov drift function $\Delta\ (L(t))$ at time t are defined as (9) and (10), respectively. Equation (9) is a function that determines how much of the current offloading workload remains at time t. Equation (10) is a function used to evaluate system stability by predicting the Lyapunov function at time $t+1$ based on the currently observable values. $\mathrm{Q}(t) = Q_1(t),\ Q_2(t),\ \ldots,\ Q_M(t)$, where i is the task number and M is the maximum number of tasks.

$$L(t) \triangleq \frac{1}{2} \sum_{i=1}^{M} \Big\{ Q_i(t)^2 \Big\} \tag{9}$$

$$\Delta\ (L(t)) \triangleq \mathbb{E}\{L(t+1) - L(t)\,|\mathrm{Q}(t)\} \tag{10}$$

To consider the maximization of the work offloading utility, we define the Lyapunov drift-plus-penalty function L as shown in (11) by adding a penalty term. V is a trade-off parameter representing the trade-off relationship.

$$L = \Delta\ (L(t)) - V\mathbb{E}\{U(t)\,|\mathrm{Q}(t)\} \tag{11}$$

Finally, the task offloading algorithm based on the Lyapunov optimization framework determines the computing server that minimizes (11) as the offloading target server. Minimizing (11) means reducing the left term, meaning system stabilization, and maximizing the right term, meaning task offloading utility simultaneously.

5.2. Derive the Optimal Solution

Theorem 1. *Lyapunov drift function upper bound.*

For any trade-off non-negative parameter V and any possible decision of offloading workload, the Lyapunov drift function has the upper bound, as shown (12).

$$L = \Delta\left(L(t)\right) - V\mathbb{E}\{U(t)|Q(t)\} \leq B + \sum_{i=1}^{M} Q_i(t)(a_i(t) - b_i(t)) - V\mathbb{E}\{U(t)|Q(t)\} \quad (12)$$

where $B = \frac{1}{2}\sum_{i=1}^{M}\left(a_i(t)^2 + b_i(t)^2\right)$.

Proof of Theorem 1. Equation (10) is expanded by (1) as follows:

$$\begin{aligned}\Delta\left(L(t)\right) &\triangleq \mathbb{E}\{L(t+1) - L(t)|Q(t)\} = \mathbb{E}\Big\{Q_i(t+1)^2 - Q_i(t)^2\Big|Q(t)\Big\} \\ &= \mathbb{E}\Big\{(max[Q_i(t) - b_i(t),\ 0] + a_i(t))^2 - Q_i(t)^2\Big|Q(t)\Big\}\end{aligned}$$

The terms inside the expectation hold the following inequality:

$$(max[Q_i(t) - b_i(t),\ 0] + a_i(t))^2 - Q_i(t)^2 \leq (Q_i(t) - b_i(t))^2 + a_i(t)^2 + 2a_i(t)Q_i(t)$$

By expanding (10) according to the above inequality relationship, the finite constant B is defined, and by substituting it into (11), the upper limit of (12) can be obtained:

$$\Delta\left(L(t)\right) \leq \frac{1}{2}\sum_{i=1}^{M}\left(a_i(t)^2 + b_i(t)^2\right) + \sum_{i=1}^{M} Q_i(t)(a_i(t) - b_i(t)) \leq B + \sum_{i=1}^{M} Q_i(t)(a_i(t) - b_i(t))$$

where $B = \frac{1}{2}\sum_{i=1}^{M}\left(a_i(t)^2 + b_i(t)^2\right)$.

The Lyapunov drift-plus-penalty function minimization problem is defined as (13). And the optimization problem F is defined as (14) by Theorem 1 and

$$min(B + \textstyle\sum_{i=1}^{M} Q_i(t)(a_i(t) - b_i(t)) - V\mathbb{E}\{U(t)|\mathrm{Q}(t)\}) \quad (13)$$

$$\begin{aligned}F:\ & max(\textstyle\sum_{i=1}^{M} Q_i(t)b_i(t) + V\mathbb{E}\{U(t)|\mathrm{Q}(t)\}), \\ & s.t. 0 \leq b_i(t) \leq min[a_i(t) + Q_i(t),\ b_i^{max}(t)]\end{aligned} \quad (14)$$

The second partial derivative of F to $b_i(t)$ is negative, so F is a convex function. In addition, as in (14), since a simultaneous inequality condition exists for $b_i(t)$, the Lagrange optimization method is used to find the optimal solution of $b_i(t)$. The Lagrange function $h(b_i(t),\ v_i)$ using the Lagrange multiplier $v_i = \{v_1,\ v_2,\ \ldots,\ v_I\}$ is defined as in (15).

$$\begin{aligned}h(b_i(t),\ v_i) &= V\Big(\textstyle\sum_{i=1}^{M} U(t)\Big) + \textstyle\sum_{i=1}^{M} Q_i(t)b_i(t) \\ &- \textstyle\sum_{i=1}^{M} v_i(b_i(t) - \min[a_i(t) + Q_i(t),\ b_i^{max}(t)])\end{aligned} \quad (15)$$

Equation (15) has necessary and sufficient conditions such as (16), (17), and (18) by the Karush-Kuhn Tucker condition [30]. By expanding (16), the optimal solution of the optimal solution of $b_i(t)$ as shown in (19).

$$\frac{\partial h(b_i(t),\ v_i)}{\partial b_i(t)} = 0 \quad (16)$$

$$v_i * \frac{\partial h(b_i(t),\ v_i)}{\partial v_i} = 0 \quad (17)$$

$$v_i \geq 0 \tag{18}$$

$$b_i(t) = \frac{V\alpha}{G \ln 2} - 1, \; where \; G = V\left(\frac{1}{BW} + \frac{1}{R_{capa}}\right) + v_i - Q_i(t) \tag{19}$$

The task offloading algorithm observes $Q_i(t)$ and R_{capa} and receives β, d_r, and BW from task analysis. The task offloading scheduler distributes each task to a queue when a stream type task request comes in and schedules tasks with real-time characteristics to be offloaded first. When determining the optimal computing server, the task offloading algorithm calculates (14) and determines the server with the maximum value. □

6. Task Offloading Algorithm

In this section, we propose a new task offloading algorithm using Lyapunov optimization, which considers the system stability and offloading utility function.

Figure 8 shows the flowchart of the proposed task offloading algorithm. The following describes the process of task offloading in detail.

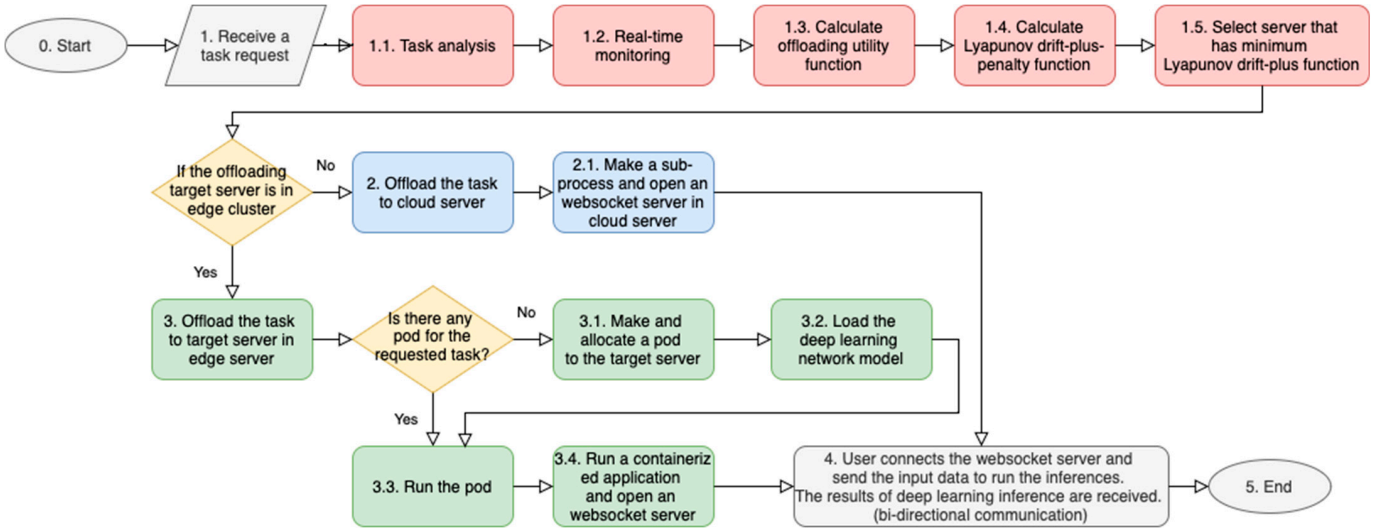

Figure 8. The proposed task offloading algorithm.

Step 1. Receive a task request: When a mobile user requests a deep learning service, the user sends a task request to the receiving server of the task offloading scheduler. In an actual mobile edge computing system, mobile users send multiple task requests simultaneously. Our task offloading algorithm is executed each time a task request is received.

Step 1.1. Task analysis: Then the task offloading scheduler analyzes the characteristics of the task. The computational complexity and the real-time characteristic are considered.

Step 1.2. Real-time monitoring: In this step, the status of the computing server, such as computing resource utilization and the amount of loads, is monitored. Some computing servers are not suitable for offloading target servers because they lack available computing resources and are expected to miss task processing deadlines due to load.

Step 1.3. Calculate offloading utility function: The offloading utility function that we defined in Section 4 is calculated. This value and the queue backlog are used to calculate the Lyapunov optimization framework.

Step 1.4. Calculate Lyapunov drift-plus-penalty function.

Step 1.5. Select the server that has minimum Lyapunov drift-plus-penalty function: The Lyapunov drift-plus-penalty function is calculated to search the optimal offloading target server. The optimal offloading target server should have the minimum value of this function. Therefore, the optimal offloading target server is selected by the task offloading scheduler.

[If the offloading target is the cloud server]

Step 2. Offload the task to the cloud server: If the target server is the cloud server, the task is processed as a containerized application in a sub-process. Firstly, the task request is offloaded to the cloud server. The cloud server makes a sub-process to run the containerized application and opens a WebSocket server to communicate with the mobile user.

[If the offloading target is in the edge cluster]

Step 3. Offload the task to target server in edge cluster: If the target server is in the Kubernetes-based edge cluster, the task is processed as a containerized application in a pod. The master node creates pods to perform tasks on worker nodes, only creating new pod for the task that have never been performed before. If the requested work has been performed before on the target worker node, the previously created pod will run to process the requested work. After a pod is allocated in the target worker node, the containerized application opens a WebSocket server to communicate with the mobile user.

Step 4. User connects to the WebSocket server and sends the input data to run the inferences. The results of deep learning inference are received. (bi-directional communication): Then the mobile user connects to the WebSocket server and sends the input data for deep learning inferences. The results of the deep learning inferences are transmitted to the user simultaneously using the bi-directional communication method.

From step 0 to step 1.5 represents the modules of task offloading determining the offloading target server. These steps consist of real-time monitoring and simple calculations of offloading utility and drift-plus-penalty functions. As in (6), the offloading utility function is asymptotically notated as the sum of a log function and linear function of given offloading workload.

As shown in (12), the Lyapunov drift-plus-penalty function has its own upper bound. The upper bound of Lyapunov drift-plus-penalty function is a linear combination of offloading workload.

From step 2 to step 5 represents how the tasks are offloaded and processed in each target server. To reduce the time complexity of this proposed algorithm, we use caching and WebSocket communication to reduce deep learning model loading time and communication latency. Also, to perform the tasks that are occurred simultaneously, the cloud server creates sub-processes in parallel, and the edge cluster manages the pod allocation.

7. Experiments

In this section, we measure the average value of queue backlog, offloading utility, and system delay over time to evaluate the performance of the proposed task offloading algorithm.

7.1. Experimental Setup

Since the proposed task offloading algorithm optimizes the trade-off relationship between system stability and offloading utility, its performance is verified by measuring the queue backlog, offloading utility, and system delay over time. Queue backlog and offloading utility are calculated and measured according to the system model defined in Section 3. System delay is measured to verify offloading performance by measuring the total delay from requesting a task to receiving the inference result.

We compare the performance of the proposed method with a random scheduling and a previous study of using Kubernetes scheduler-based offloading [31]. The random scheduling is a basic offloading method that randomly assigns offloading workloads to

servers. In [31], the authors propose a Kubernetes scheduler-based offloading method that considers a computing server's limit and GPU capacity when processing neural network services in mobile edge computing.

The workload samples for all experiments are created using dbbench, a workload generator program. To evaluate the performance in various scenarios, we generate the workload samples by controlling the generation frequency as 40 input data sets/s and 80 input data sets/s. The types of requested tasks are all randomly generated. Also, to consider scenarios that may occur in real autonomous driving environment, we generate workload samples with a high density of specific task requests. We average the results by repeating the experiments three times in the same environment.

The mobile edge computing environment introduced in Section 4 is used for experiments. CARLA, a virtual urban driving simulator, and ROS are used to build an autonomous driving environment and to transmit data between virtual vehicles and cities [32]. The hardware and software specifications are shown in Table 6. This paper uses the benchmarking result of Nvidia Jetson series embedded devices to build an edge cluster using small devices with various computing resources [33]. As a result, Nvidia Jetson AGX Xavier, TX2, and Nano are selected, and their specifications are shown in Table 7. The cloud server consists of a front server and the sub-processes by utilizing AWS deep learning instance to implement a physically distant server with abundant computing resources. The front server receives offloading requests and a sub-process that provides deep learning inference service.

Table 6. Environmental Setup of the Mobile User.

Hardware	Software
CPU: Intel Core i5-7500 GPU: GeForce GTX 1080 8G RAM: 32 GB	CARLA Simulator version: 0.9.10 ROS version: Noetic

Table 7. Environmental Setup of the Edge Cluster.

Xavier	TX2	Nano
RAM: 16 GB LPDDR4	RAM: 8 GB LPDDR4	RAM: 4 GB LPDDR4
CPU: 8-core ARMv8.2 8 MB L2 + 4 MB L3	CPU: 4-core ARM A57 2 MB L2 + HMP Dual Denver 2/2 MB L2	CPU: Quad-core ARM cortex-57
GPU: 512-core Volta with Tensor Core	GPU: 256-core Pascal	GPU: 128-core Maxell

7.2. System Stability Evaluation

Figure 9 shows a graph of queue backlog over time when the workload generating frequency is 40 input data sets/s. The proposed task offloading results show that the queue load increases because tasks are accumulated in the queue, and it takes time to load the neural network model for several seconds. However, after that, it remains stable and prevents divergence. Random scheduling fails to account for dynamically changing computing resources, causing queue backlog to diverge within seconds. In particular, in the case of Kubernetes scheduler-based offloading, it cannot efficiently allocate the concurrently requested tasks, so pending pods occur. Pending is a state in which tasks are not processed until the worker nodes' availability is confirmed, so the system is unstable, and the deadline cannot be guaranteed.

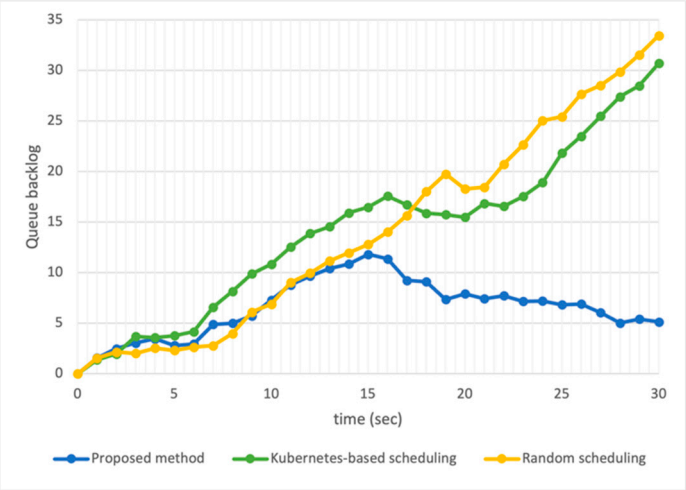

Figure 9. Queue backlog over time (40 input data sets/s).

7.3. Offloading Utility Evaluation

The workload samples for offloading utility evaluation are same as that are used in the system stability evaluation. As shown in Figure 10, the offloading utility value periodically increases and then decreases over time. It is because we set the trade-off parameter V to give more importance to system stability. System designer can control V to determine the importance of queue backlog and offloading utility.

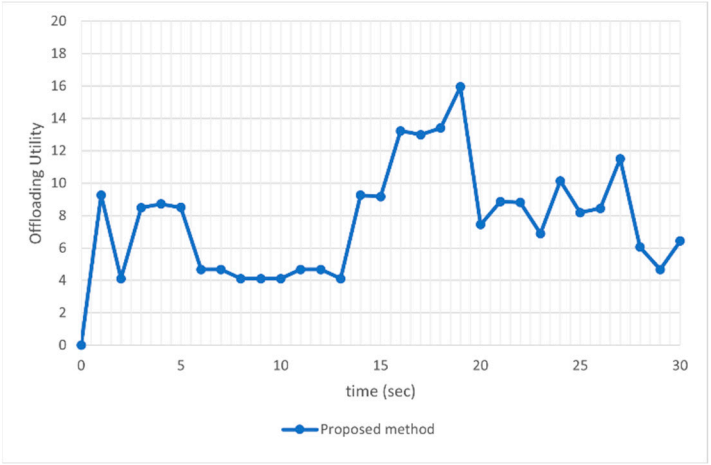

Figure 10. Offloading utility over time.

7.4. System Delay Evaluation

Deep learning services used in autonomous driving environments must complete tasks within the deadline for user safety. In the case of object detection, since an object on the front side must be detected in real-time during autonomous driving, the task must be completed within 0.6 s, and driver profiling and image recognition must also be provided within 1 s. End-to-end delay is the total delay from when a user requests a task to receive the result of the task, i.e., the time it takes to complete. End-to-end delay is the sum of queuing delay, pod allocation delay, transmission delay, and execution delay.

Table 8 shows the average value of the end-to-end delay for 20 experiments. In the case of the proposed method, object detection with real-time characteristics takes 0.4231 s for each request, and driver profiling and image recognition complete the task within 0.7095 s and 0.9017 s, respectively. Therefore, the proposed task offloading algorithm confirms that deep learning applications can be efficiently utilized in real-time in an autonomous driving

environment. About Kubernetes-based scheduling and random scheduling, the end-to-end time of all the tasks are huge because of the pending issue.

Table 8. End-to-end Delay of each Deep Learning Service During Task Offloading.

	Driver Profiling	Image Recognition	Object Detection
Proposed method	0.7095 s	0.9017 s	0.4231 s
Kubernetes-based scheduling	4.6322 s	8.6410 s	5.3642 s
Random scheduling	6.3836 s	7.3792 s	14.3258 s

8. Conclusions

This paper proposed a task offloading algorithm for deep learning services to keep the system stable and minimize task processing delay in an autonomous driving environment. We design mobile edge computing for the autonomous driving environment and a real-time server monitoring system to efficiently utilize distributed computing resources. For simulating the real-life cases, embedded devices are used by orchestrating the hardware resources using Kubernetes. To achieve the purpose of system stability and delay minimization, we define the task offloading utility and analyze the characteristics of deep learning applications. As of being popular applications used in autonomous driving, this research selected driver profiling, image recognition and object detection for testing purposes. To solve the optimization problem, the task offloading steps are systematically formulated using Lyapunov optimization and Lagrange optimization framework. Further evaluations were performed to prove the advantage of proposed system over existing approaches. The experimental results confirm that the queue backlog is stable and does not diverge. Object detection application is completed within an average of 0.4231 s, driver profiling within an average of 0.7095 s, and image recognition within an average of 0.9017 s. We ensure that the proposed task offloading algorithm enables deep learning applications to be processed within the deadline and maintains the system stably in an autonomous driving environment.

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

Funding: This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) under Grant NRF-2021R1F1A1050750 and in part by the Inha university research grant.

Data Availability Statement: Data is not available for this research.

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

References

1. Hussain, R.; Zeadally, S. Autonomous cars: Research results, issues, and future challenges. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 1275–1313. [CrossRef]
2. Liu, L.; Lu, S.; Zhong, R.; Wu, B.; Yao, Y.; Zhang, Q.; Shi, W. Computing systems for autonomous driving: State of the art and challenges. *IEEE Internet Things J.* **2020**, *8*, 6469–6486. [CrossRef]
3. Liu, L.; Zhao, M.; Yu, M.; Jan, M.A.; Lan, D.; Taherkordi, A. Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 2169–2182. [CrossRef]
4. Ahmed, M.; Raza, S.; Mirza, M.A.; Aziz, A.; Khan, M.A.; Khan, W.U.; Li, J.; Han, Z. A Survey on Vehicular Task Offloading: Classification, Issues, and Challenges. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 4135–4162. [CrossRef]
5. Stilgoe, J. Machine learning, social learning and the governance of self-driving cars. *Soc. Stud. Sci.* **2018**, *48*, 25–56. [CrossRef]
6. Chen, M.; Hao, Y. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 587–597. [CrossRef]
7. Alameddine, H.A.; Sharafeddine, S.; Sebbah, S.; Ayoubi, S.; Assi, C. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 668–682. [CrossRef]

8. Li, Y.; Xia, S.; Zheng, M.; Cao, B.; Liu, Q. Lyapunov optimization based trade-off policy for mobile cloud offloading in heterogeneous wireless networks. *IEEE Trans. Cloud Comput.* **2019**, *10*, 491–505. [CrossRef]
9. Liu, Q.; Chen, Z.; Wu, J.; Deng, Y.; Liu, K.; Wang, L. An efficient task scheduling strategy utilizing mobile edge computing in autonomous driving environment. *Electronics* **2019**, *8*, 1221. [CrossRef]
10. Zhang, J.; Hu, X.; Ning, Z.; Ngai, E.C.-H.; Zhou, L.; Wei, J.; Cheng, J.; Hu, B. Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks. *IEEE Internet Things J.* **2017**, *5*, 2633–2645. [CrossRef]
11. Liu, S.; Wang, J.; Wang, Z.; Yu, B.; Hu, W.; Liu, Y.; Tang, J.; Song, S.L.; Liu, C.; Hu, Y. Brief Industry Paper: The Necessity of Adaptive Data Fusion in Infrastructure-Augmented Autonomous Driving System. In Proceedings of the 2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS), Milano, Italy, 4–6 May 2022; pp. 293–296. [CrossRef]
12. Terefe, M.B.; Lee, H.; Heo, N.; Fox, G.C.; Oh, S. Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing. *Pervasive Mob. Comput.* **2016**, *27*, 75–89. [CrossRef]
13. Yang, G.; Hou, L.; He, X.; He, D.; Chan, S.; Guizani, M. Offloading time optimization via Markov decision process in mobile-edge computing. *IEEE Internet Things J.* **2020**, *8*, 2483–2493. [CrossRef]
14. Islam, A.; Debnath, A.; Ghose, M.; Chakraborty, S. A survey on task offloading in multi-access edge computing. *J. Syst. Arch.* **2021**, *118*, 102225. [CrossRef]
15. Li, C.; Xia, J.; Liu, F.; Li, D.; Fan, L.; Karagiannidis, G.K.; Nallanathan, A. Dynamic offloading for multiuser muti-CAP MEC networks: A deep reinforcement learning approach. *IEEE Trans. Veh. Technol.* **2021**, *70*, 2922–2927. [CrossRef]
16. Neely, M.J. Stochastic network optimization with application to communication and queueing systems. *Synth. Lect. Commun. Netw.* **2010**, *3*, 1–211.
17. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018.
18. Ullah, S.; Kim, D.-H. Lightweight driver behavior identification model with sparse learning on in-vehicle can-bus sensor data. *Sensors* **2020**, *20*, 5030. [CrossRef] [PubMed]
19. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018.
20. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
21. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In Proceedings of the Conference on Robot Learning, Mountain View, CA, USA, 13–15 November 2017.
22. Lee, H.-G.; Kim, D.-H. AUTOWARE linkage and MPC control using virtual simulator CARLA. *KINGPC* **2021**, 363–366.
23. Felter, W.; Ferreira, A.; Rajamony, R.; Rubio, J. An updated performance comparison of virtual machines and linux containers. In Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 29–31 March 2015.
24. Hoque, S.; De Brito, M.S.; Willner, A.; Keil, O.; Magedanz, T. Towards container orchestration in fog computing infrastructures. In Proceedings of the 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, Italy, 4–8 July 2017; Volume 2.
25. Zhong, Z.; Buyya, R. A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources. *ACM Trans. Internet Technol.* **2020**, *20*, 1–24. [CrossRef]
26. Fu, Y.; Zhang, S.; Terrero, J.; Mao, Y.; Liu, G.; Li, S.; Tao, D. Progress-based container scheduling for short-lived applications in a kubernetes cluster. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019.
27. Casalicchio, E. Container Orchestration: A Survey. In *Systems Modeling: Methodologies and Tools*; Springer: Cham, Switzerland, 2018; pp. 221–235.
28. Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *239*, 2.
29. Shephard, R.W.; Färe, R. The law of diminishing returns. In *Production Theory*; Springer: Berlin/Heidelberg, Germany, 1974; pp. 287–318.
30. Boyd, S.P.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004.
31. Kim, J.; Ullah, S.; Kim, D.-H. GPU-based embedded edge server configuration and offloading for a neural network service. *J. Supercomput.* **2021**, *77*, 8593–8621. [CrossRef]
32. Lee, H.-G.; Kang, D.-H.; Kim, D.-H. Human–Machine Interaction in Driving Assistant Systems for Semi-Autonomous Driving Vehicles. *Electronics* **2021**, *10*, 2405. [CrossRef]
33. Ullah, S.; Kim, D.-H. Benchmarking Jetson Platform for 3D Point-Cloud and Hyper-Spectral Image Classification. In Proceedings of the 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), Busan, Republic of Korea, 19–22 February 2020; pp. 477–482.