ECSeg: Edge-Cloud Switched Image Segmentation for Autonomous Vehicles

Siyuan Zhou, Duc Van Le, Rui Tan
College of Computing and Data Science, Nanyang Technological University, Singapore

***Abstract*—Existing autonomous vehicles have not utilized the cloud computing for execution of their deep learning-based driving tasks due to the long vehicle-to-cloud communication latency. Meanwhile, the vehicles are in general equipped with the resource-constrained edge computing devices which may be unable to execute the compute-intensive deep learning models in real time. The increasing data transmission speed of the commercial mobile networks sheds light upon the feasibility of using the cloud computing for autonomous driving. Our city-scale real-world measurements show that the vehicles can partially use the cloud computing via the fifth generation (5G) mobile network with the low data transmission latency. In this paper, we present the design and implementation of ECSeg, an edge-cloud switched image segmentation system that dynamically switches between the edge and cloud for executing the deep learning-based semantic segmentation models to understand the vehicle's visual scenes in real time. The switching decision-making is challenging due to the intricate interdependencies among various factors including the dynamic wireless channel condition, vehicle's movement and visual scene change. To this end, we employ deep reinforcement learning to learn an optimal switching policy. Extensive evaluation based on both real-world experiments and trace-driven simulations demonstrates that ECSeg achieves superior image segmentation accuracy for autonomous vehicles, compared with four baseline approaches.**

I. Introduction

Autonomous vehicles (AVs) have substantial potential to mitigate traffic congestion, enhance road safety, and curtail carbon emissions. Deep learning (DL) has been increasingly employed for various driving tasks of the AVs. For example, the DL models [1]–[3] can be used for the vehicles to understand their visual driving scenes correctly, facilitating the safe driving navigation and accurate collision avoidance. To avoid the long latency and privacy issues of data transmission, the commercial AV platforms (e.g., Apollo [4]) are often equipped with the resource-limited edge computing devices to directly execute the DL-based autonomous driving tasks on the vehicles. Meanwhile, the execution of deep models often requires high demand on computing resources. Thus, current AV design strategies adopt customized lightweight, on-board deep models [5] which can be executed by the edge devices in real time to achieve autonomous driving. This design choice compromises the accuracy of the deep models. A possible approach to address this problem is to increase the computing capabilities of the edge devices which allow implementation of complex deep models with high accuracy. However, the powerful computing devices are energy-intensive, which reduces the vehicle's battery system lifetime. It is also challenging for the vehicle's heat dissipation system to handle the tremendous amount of heat dissipated by the energy-intensive computing devices [6].

This research is supported by Singapore Ministry of Education under its AcRF Tier-1 grant RT14/22.

In this paper, we investigate the feasibility of using the cloud computing for executing the DL-based autonomous driving tasks. In particular, the cloud servers can provide the AVs with sufficient computing capabilities without power limitation and heat dissipation issues. To address the privacy concerns, the data encryption approaches [7] can be implemented to secure the AV's sensitive information before transmitting the data to the cloud server. Meanwhile, the increasing data transmission speed of the commercial mobile networks sheds light upon the opportunity of significantly reducing the vehicle-to-cloud communication latency. Our city-scale real-world measurements show that the fifth generation (5G) mobile network can provide an average round-trip time (RTT) latency of 100 ms for transmitting the 500-KB data packets from the moving vehicles to the remote cloud servers. Moreover, the total power usage of the vehicular 5G communication and computing operations constitutes less than 1% of the vehicle's total mechanical energy usage. Thus, we conjecture that clouds connected via a mobile network can bring benefits to the AVs.

To further demonstrate the benefits of the cloud computing for AVs, in this paper, we present the design of an edge-cloud switched image segmentation system, called *ECSeg* which aims to provide pixel-level understanding of the vehicle's visual driving scenes in real time. Specifically, ECSeg switches between different deep models to obtain the segmentation result of each image frame captured by the AV's camera before a certain deadline (e.g., the time when the next image frame is captured). To achieve the goal, ECSeg has the following two processing options. First, *edge processing option* executes a lightweight convolutional neural network (CNN) model locally on the AV's edge computing device to obtain the segmentation results of the image frames. Second, *cloud processing option* compresses the raw images and transmits them to a cloud server via the mobile network. Then, the cloud server executes an advanced CNN model to process the images and sends the results back to the vehicle.

Among the above two processing options of ECSeg, the edge processing option can always provide the segmentation result of the captured image frame in real time, however it has low segmentation accuracy due to the use of the lightweight CNN model. With the execution of the advanced CNN model, the cloud processing option can provide the segmentation result with higher accuracy to the vehicle. However, due to the cloud data transmission latency, the vehicle may not always obtain the segmentation result of a transmitted image frame before the image's processing deadline. To mitigate the long cloud latency issue, the cloud processing option allows the vehicle to only spend a certain time period to wait for the cloud result of the current image frame (i.e., current driving scene). We define the latest frame among the previous frames whose cloud segmentation results have already arrived at the vehicle as the source frame. If the vehicle does not receive the cloud result of the current frame after the waiting period, it will use the segmentation result of the source frame to interpolate the result of the current frame. Specifically, we develop an optical flow-based approach [8] to propagate the segmentation result of the source frame to the current frame. We also introduce a new metric, called delay-mitigated mean intersection over union (mIoU) to assess accuracy of the image segmentation results obtained by our developed propagation approach.

The delay-mitigated mIoU may be affected by changes in the visual content between the source and current frames. As the vehicle moves, the condition of the mobile network connection may vary due to base station switching and radio signal blockage [9]. Poor mobile network connection can cause long vehicle-to-cloud communication latency. As a result, the source frame can be far away by multiple image intervals from the current frame, leading to significant content changes between these two frames. Such changes reduce the delay-mitigated mIoU of the current frame. For example, when a new object that does not appear in the source frame enters the current frame, the propagation approach fails to obtain the segmentation result of this new object. Thus, the segmentation accuracy of the cloud processing option can be lower than that of the edge processing option. Therefore, ECSeg needs to dynamically switch between edge and cloud processing options such that the images can be processed within the deadlines while maximizing the image segmentation accuracy.

The switching decision-making is challenging due to the intricate interdependencies among various factors including the dynamic mobile network channel condition, vehicle's movement and visual driving scene change. To this end, we employ deep reinforcement learning (DRL) to learn a good long-term switching policy. However, the typical online training of the DRL requires excessive time to converge, which may raise safety concerns for autonomous driving. Moreover, there is a lack of image ground truth for determining the immediate rewards during the online training phase. To address these challenges, we adopt an offline training approach which utilizes real-world data traces to train the DRL agent. Finally, the trained agent is deployed for making the edge-cloud switching decisions at runtime.

We extensively evaluate the performance of ECSeg in both real-world experiments and trace-driven simulations. We also compare our proposed DRL-based switching approach with four baselines that incorporate efficient DL and knowledge distillation approaches from existing research. The evaluation results show that ECSeg can outperform the baselines with up to 48.8% in terms of mIoU (i.e., segmentation accuracy). Our main contributions can be summarized as follows:

- We design and implement an edge-cloud switched image segmentation pipeline and conduct large-scale measurements to evaluate the benefits of using the cloud computing for AVs. Our design and experimental results may be useful for the development of other edge-cloud collaboration pipelines for AVs.
- We propose a new metric called *delay-mitigated mIoU* for the cloud-based image segmentation in AVs. We formulate the edge-cloud switching problem and adopt a DRL-based approach to build the decision policy.
- We conduct extensive evaluations on real-world testbeds to assess the effectiveness of our proposed approach. Our results indicate that our approach outperforms the baseline methods.

The remainder of this paper is organized as follows. §II reviews related work. §III presents the measurement study. §IV overviews ECSeg's design. §V describes the cloud processing option. §VI presents the DRL-based switching approach. §VII presents the evaluation results. §VIII concludes this paper.

II. Related Work

In this section, we review the existing studies on the edge-based AVs, cloud-enabled sensing system, and RL-based control policy.

■ **Edge-based AVs:** Edge platforms have been adopted to execute the DL-based driving tasks in the AVs [2], [3], [10]. For instance, the study in [2] employs a CNN model with early downsampling and smaller convolutional filters to identify road lanes in real time on a vehicle platform. The authors in [10] adopt a sparse scaling factor algorithm to identify and prune less important channels and weights of CNN models for detecting vehicles and traffic signs, tailored for deployment on vehicle-mounted computing platforms. Additionally, the study in [3] proposes a knowledge transfer method to train binary weight neural networks using a full-precision model for object detection in autonomous driving. These customized CNN models for resource-limited devices often compromise accuracy, while we use high-accuracy advanced CNN models supported by a cloud server.

■ **Cloud-enabled sensing system:** Several existing cloud-enabled systems [11]–[13] have been employed to enhance the task performance on resource-limited devices. For example, the study in [11] offloads tasks to the cloud server based on the task buffer queuing state, transmission state, and local processing state to minimize task latency and reduce power consumption in mobile devices. ACCUMO [13] uses a large CNN model on the cloud server and a lightweight local tracker to adjust cloud results for a real-time augmented reality task. Based on this, it employs model predictive control, using predicted accuracy, to determine which tasks to offload to the cloud for multiple tasks, optimizing overall accuracy. The above approaches always offload tasks to cloud servers, which can result in long latency when the wireless condition is poor.

In this paper, we develop an edge-cloud switched method that shifts to edge processing when wireless condition is poor.

Similar to our work, DeepDecision [12] determines whether to process images on the cloud using a large object detection model or locally at the edge, based on network conditions and hardware limitations, with the main objective of maximizing the accuracy and frame rate. However, DeepDecision cannot be directly applied to our AV scenario because it does not consider the processing deadline for each frame. Additionally, it does not account for how changes in the driving scene affect the accuracy of cloud model. Our work aims to adopt ECSeg, an edge-cloud switched method to maximize the mIoU of image segmentation for AVs meeting the frame processing deadline. We compare ECSeg with ACCUMO and DeepDecision in §VII.

■ **RL-based control policy:** Reinforcement learning (RL) has been applied to decide the task offloading in the edge/cloud computing systems [14]–[16]. The study [14] uses an online DRL agent to decide whether to offload tasks to the cloud server from multiple users, based on the users' queue and channel states, to optimize the age of information of the task. FEAT [15] uses DRL to decide task offloading to edge servers, based on task size, bandwidth, and hardware observations, reducing latency and mobile device energy consumption, while employing a steerer neural network to switch the DRL agent based on changes in observation distribution. EFCam [16] adopts the RL agent to adapt the wireless camera configuration to maintain the high performance of visual sensing. Our work shares a similar DRL-based control approach for a switching strategy to improve segmentation accuracy in AVs while considering the impacts of dynamic cloud latency and driving scene changes.

III. Measurement study

In this section, we conduct a set of city-scale measurement experiments to study the vehicle-to-cloud communication latency in various urban areas. Then, we investigate the execution accuracy and latency of the state-of-the-art deep image segmentation models on the edge and cloud platforms. The results provide insights to guide the design of our edge-cloud switched segmentation approach.

A. *Vehicle-to-Cloud Communication Latency*

We conduct a set of real-world experiments to measure the vehicle-to-cloud communication latency in various city areas. Specifically, we build a vehicular communication testbed which consists of a smartphone with a 5G communication service mounted on a vehicle. We use the Python socket library to implement a client-server application that allows a smartphone to transmit the data to a remote cloud server using the TCP/IP protocol via the 5G mobile network. In each experiment, the smartphone continuously transmits 500-KB application data packets to a Google's cloud server while traveling on a vehicle with the movement speed up to 50 km/h. Upon receiving each data packet, the cloud server sends a 1-byte acknowledgment packet back to the smartphone. We measure the round-trip time (RTT) of all transmitted data packets. The RTT of a data packet is defined as the time that it takes to receive the acknowledgment packet after the smartphone transmits the data packet.

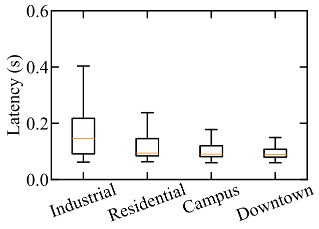


Fig. 1. Vehicle-to-cloud latency.

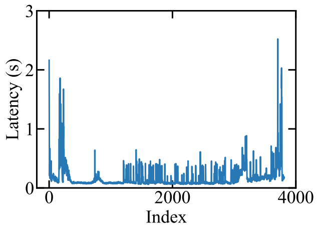


Fig. 2. Campus latency trace.

Fig. 1 shows box plots for the distributions of the RTTs of the data packets that are transmitted from the smartphone to the cloud server over a duration of 1.5 hours in four typical city areas: university campus, residential, downtown, and industrial areas. From Fig. 1, all four areas have an average RTT within a few hundred milliseconds with fluctuations. The downtown area has the shortest average RTT of about 112 ms, while the industrial area has the longest average of about 182 ms with significant fluctuations. The reason is the downtown area has a high density of 5G base stations, which reduces the data travel time between the smartphone and base station. In contrast, the industrial area may have fewer base stations. Fig. 2 presents the RTT trace of the data packets in the university campus area. As the vehicle moves, the RTT fluctuates and reaches up to two seconds. This communication latency variability may be due to the switching of 5G base stations and signal blockage caused by buildings or other vehicles.

B. *Image Segmentation Latency and Accuracy*

We conduct profiling experiments to investigate the execution accuracy and latency of various deep learning-based image segmentation models on both the edge and cloud platforms. Specifically, we use an NVIDIA Jetson Orin [17] unit with a 32-GB DRAM as the AV's edge computing platform. A workstation equipped with an RTX 8000 GPU serves as the cloud platform. We implement eight image segmentation models with different sizes, including three lightweight models: ESPNet [5], Fast-SCNN [18], and DeepLabv3+ [19], as well as five large models: PSPNet-ResNet50 [20], Mask2Former-swin-s [21], Mask2Former-swin-b [21], InternImage-XL [22]. and InternImage-H [22]. We use the images sized $2048 \times 1024 \times 3$ pixels from the Cityscapes dataset [23] to evaluate the segmentation accuracy and latency of the implemented models.

Table I presents the number of parameters, memory size, mIoU (i.e., accuracy) and latency of the segmentation models executed on the edge and cloud platforms. The cloud latency is the end-to-end latency for image data transmission and model execution. We use the average RTT in the university campus area (c.f. §III-A) as the transmission latency of each image. The mIoU and latency numbers are the average values of the models over 1,000 testing images. From Table I, the image segmentation accuracy and latency in general increase with the

Tab. I
MODEL EXECUTION LATENCY.

Model	Params (M)	Size (MB)	mIoU	Latency (ms)	
				Edge	Cloud
ESPNet	0.36	1.37	0.505	30	139
Fast-SCNN	1.45	5.53	0.682	65	136
DeepLabv3+-ResNet18	12	47.45	0.743	170	190
PSPNet-ResNet50	49	186.81	0.785	720	360
Mask2Former-swin-s	69	262.15	0.803	890	430
Mask2Former-swin-b	107	407.71	0.824	940	530
InternImage-XL	368	1402.78	0.836	1050	660
InternImage-H	1080	4119.87	0.841	N.A[1]	5270

[1] The edge platform runs out of memory during inference.

model complexity and size. Without the need of transmitting the images to the cloud, the edge platform can execute the lightweight models and provide the segmentation results with shorter latencies, compared with the cloud platform. On the other hand, the cloud platform can provide the segmentation results of large models with shorter end-to-end latency, even when including the cloud communication latency.

C. Key Findings

From the above measurements, we can see that the cloud server can execute the large models to provide higher mIoU and shorter latency, compared with the edge platform. However, cloud transmission latency fluctuates with vehicle movement, and the long transmission delays result in the long end-to-end cloud latency. To prevent low mIoU caused by cloud latency, we propose a strategy of switching between edge and cloud processing options based on factors discussed in §V that impact the cloud results' mIoU.

IV. DESIGN OF ECSEG

We consider an AV that uses a camera system to periodically capture its driving scene at every sampling interval, denoted by T. ECSeg is designed to execute deep learning-based semantic segmentation models to obtain the segmentation result of each image frame before the time when the next frame is captured. The image segmentation results provide pixel-level understanding of the AV's driving scenes which can be used as inputs for the autonomous driving pipelines [24], [25] to achieve accurate trajectory planning and safe navigation. Fig. 3 overviews the design of ECSeg which has two options: edge and cloud processing options for image segmentation as follows.

■ **Edge processing option:** This option executes a segmentation model locally to obtain the segmentation results of the captured image frames on the AV's edge platform. Given the limited computing resources of the edge platform, ECSeg employs a lightweight CNN-based image segmentation model as the local model such that the image segmentation result of each image can be always obtained before its deadline.

■ **Cloud processing option:** This option follows a streaming mode to continuously transmit the image frames from the vehicle to the cloud server via a mobile network. To reduce the communication overheads, we implement a JPEG approach to compress each image before transmitting it to the cloud server.

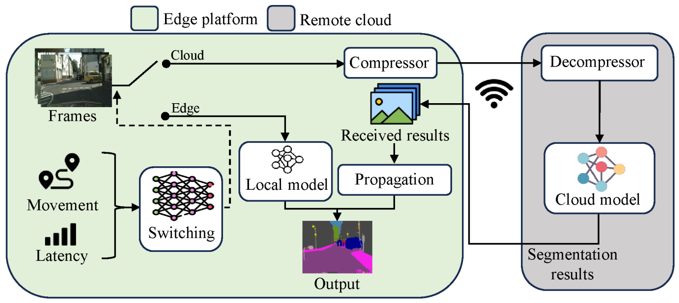


Fig. 3. Design overview of ECSeg.

Upon receiving the image data, the cloud server employs a JPEG decompressor to reconstruct the original image. Then, the cloud server executes the cloud model to process the reconstructed image. To achieve high image segmentation accuracy, an advanced CNN-based model with large size is implemented as the cloud model in the cloud server. Finally, the cloud image segmentation result is sent back to the vehicle.

Due to the long vehicle-to-cloud communication latency, the cloud segmentation result of an image frame may not arrive at the vehicle before the deadline. Thus, we also develop a propagation approach which uses the received cloud result of a previous frame as input to obtain the segmentation result for the current frame. The detailed design of our segmentation result propagation approach will be presented in §V.

■ **Switching:** The edge processing option can always provide the image segmentation result within the deadline. However, it suffers from low segmentation accuracy due to the use of the lightweight model. In contrast, the cloud processing option can execute an advanced model to achieve high accuracy, but has high latency uncertainty due to the dynamic vehicle-to-cloud communication latency. Due to the vehicle movement and poor wireless channel condition, the communication latency can be long, which causes the cloud segmentation results to become stale, decreasing the segmentation accuracy. To maximize the segmentation accuracy, at the edge platform, we implement a DRL-based controller which aims to dynamically switch between the edge and cloud processing options in response to changes of the communication latency and driving scene. In §VI, we formally formulate the switching problem and present our DRL-based solution.

V. CLOUD PROCESSING OPTION

A. Segmentation Result Propagation

Let $x_1, x_2, x_3, \ldots$ denote a sequence of images captured by the vehicle's camera system at every fixed interval of T. We also denote t_i as the time when the image x_i is captured and then transmitted to the cloud server. The processing deadline of x_i is $t_i + T$ (i.e., the arrival time of the next image x_{i+1}). Due to the dynamic vehicle-to-cloud communication latency, the vehicle may not receive the cloud segmentation result of x_i before the deadline $t_i + T$. To address this deadline missing issue, the vehicle only waits for the cloud result of x_i for a certain period, denoted by T_w, after t_i. If the vehicle does not receive the cloud result of x_i within the waiting period T_w,

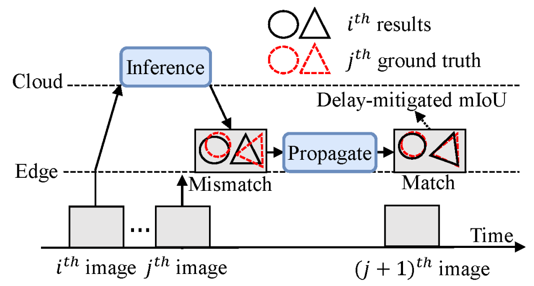


Fig. 4. Delay-mitigated mIoU.

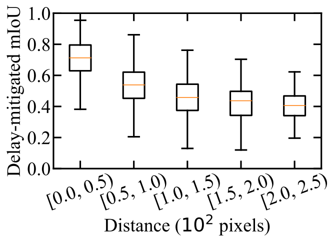


Fig. 5. Accuracy vs. image distance.

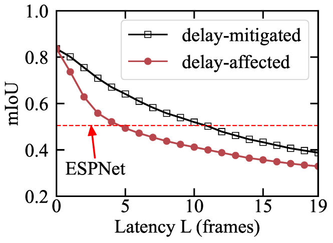


Fig. 6. Accuracy vs. Latency.

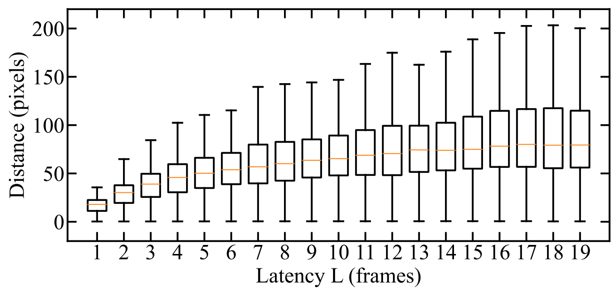


Fig. 7. Latency vs. image distance.

we employ a propagation approach to obtain the segmentation result of x_i based on the cloud segmentation result of the latest frame, denoted by x_k, among the previous frames of x_j $(j < i)$ whose cloud result has already arrived at the vehicle. Let T_p denote the fixed execution latency of the propagation. Then, the waiting period is calculated as $T_w = T - T_p$.

Now, we describe our approach to propagate the cloud segmentation result of x_k to the current frame x_i where $k < i$. The images x_k and x_i have the same size in terms of the number of pixels, denoted by M. Let $\mathcal{P}_k = \{P_{k,1}, \ldots, P_{k,M}\}$ and $\mathcal{P}_i = \{P_{i,1}, \ldots, P_{i,M}\}$ denote the sets of pixels of x_k and x_i, respectively. We adopt a computer vision technique, called optical flow which aims to determine the movement velocity of the pixels from x_k to x_i. Specifically, it derives the movement vectors, denoted by $F_{k\to i} = \{F_1, \ldots, F_M\}$ which are used to locate the pixels $\mathcal{P}_k$ of x_k within x_i. In this work, we adopt the dense inverse search-based approach proposed in [8] to derive $F_{k\to i}$ efficiently. Two pixels $P_{k,l} \in \mathcal{P}_k$ and $P_{i,h} \in \mathcal{P}_i$ share the same segmentation class label if $P_{i,h} = P_{k,l} + F_h$. Following this mapping method, the cloud segmentation class labels of M pixels in the previous image x_k are propagated to M pixels of the current image x_i. As a result, we can obtain the cloud segmentation result of x_i before its deadline.

B. Delay-mitigated mIoU

1) Definition: To assess the accuracy of the cloud segmentation results obtained via propagation, we introduce a segmentation accuracy metric, called delay-mitigated mean intersection over union (mIoU). Fig. 4 shows an example of delay-mitigated mIoU, where the i^{th} image is transmitted to the cloud and its cloud result arrives at the vehicle within the interval between the j^{th} and $(j+1)^{th}$ images. Due to the vehicle's movement, the scene captured in the j^{th} image may shift from the i^{th} image, causing the received cloud result to mismatch with the j^{th} image's ground truth. We adopt the propagation method, discussed in §V-A, to mitigate the mismatch. As a result, the delay-mitigated mIoU for the j^{th} image is the mIoU between the propagated i^{th} cloud result and the ground truth of the j^{th} image. When the pixel content of the i^{th} image differs significantly from that of the j^{th} image, the delay-mitigated mIoU may decrease.

2) Impact of image content changes: We conduct experiments to investigate the impact of content changes between the i^{th} and j^{th} images, referred to as image distance, on delay-mitigated mIoU. Specifically, we use the average Euclidean distance of each pixel in the movement vectors between two segmented images by InternImage [22] to indicate the image distance. The large image distance indicates fast-changing scenes between two images, such as more objects entering and exiting the scene, as well as rapid changes within the scene itself. We divide image distances into five ranges: $[0, 50)$, $[50, 100)$, $[100, 150)$, $[150, 200)$, and $[200, 250)$. We select 267 images from the Cityscapes dataset [23], which provides $2048 \times 1024 \times 3$ resolution images with segmentation ground truth, to compute the delay-mitigated mIoU within these ranges. Fig. 5 shows the distribution of delay-mitigated mIoU with different image distances. The mIoU values range from 0.2 to 0.95. With the image distance increasing, the mIoU decreases. This decline is due to larger distances, where captured scene changes may include objects entering and exiting. Thus, the propagation cannot mitigate the mismatch.

3) Impact of latency: We further conduct experiments to investigate the impact of end-to-end cloud latency on the delay-mitigated mIoU of received cloud results. Specifically, we select 267 images to transmit to the cloud server, where the InternImage serves as the cloud model to process these images. We denote end-to-end cloud latency as L, measured in the number of image intervals T, where $0 \le L \le 19$. $L = 0$ means that the cloud results can be received within the interval of the transmitted image. For each transmitted image, we compute the delay-mitigated mIoU after propagation for L values ranging from 0 to 19, resulting in 5,340 values. The delay-mitigated mIoU reflects the effectiveness of mismatch mitigation through propagation. Additionally, we also compute the mIoU between cloud results without propagation and the ground truth over the same L range (0 to 19), referred to as delay-affected mIoU. Fig. 6 shows the average delay-affected and delay-mitigated mIoU across different L values. The delay-mitigated mIoU is always higher than the delay-affected mIoU, ranging from 0.4 to 0.82. The results show that the propagation can mitigate the impacts of cloud latency. Both mIoU values decrease with L. Fig. 7 shows the distribution of image distance for various L values. Larger L values correspond to a larger average image distance and a wider range of image distances. This is attributed to the increased likelihood of objects entering and exiting the scene between frames as latency increases. The results show that longer cloud latency may result in a lower delay-mitigated mIoU.

VI. DRL-BASED SWITCHING

A. Problem Statement

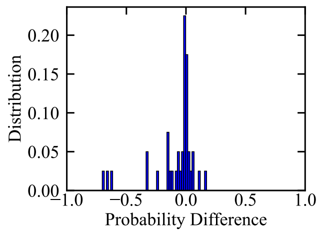
1) Optimization formulation: Time is divided into intervals with identical duration of $\tau \geq T$ seconds, which is referred to as switching period. At the beginning of switching period, called time step, the ECSeg selects the edge or cloud processing options for image segmentation in response to the changes of two exogenous stochastic factors, including the time-varying cloud transmission latency, denoted by $\eta(t)$, and the vehicle's driving scene variation, denoted by $\xi(t)$. Denote $\eta_k = \eta(k\tau)$ and $\xi_k = \xi(k\tau)$, where $k \in \mathbb{Z}_{\geq 0}$. Denote by $\eta_{t=k\tau}^{(k+1)\tau}$ and $\xi_{t=k\tau}^{(k+1)\tau}$ the trace of $\eta(t)$ and $\xi(t)$ when $t \in [k\tau, (k+1)\tau]$. At the k^{th} time step, we let $\pi(\xi_k, \eta_k, \ldots, \xi_0, \eta_0)$ denote the policy that determines a switching decision, denoted by ω_k, based on the historical measurements of $(\xi_k, \eta_k, \ldots, \xi_0, \eta_0)$. The ω_k represents the decisions, including the edge and cloud processing options, which jointly affect the delay-mitigated accuracy during the switching period. For a time horizon of K switching periods, the switching aims to solve the policy optimization problem:

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} \; \mathbb{E}_{\xi,\eta} \left[\frac{1}{K} \sum_{k=0}^{K-1} A_k \left(\omega_k, \xi_{t=k\tau}^{(k+1)\tau}, \eta_{t=k\tau}^{(k+1)\tau} \right) \right], \quad (1)$$

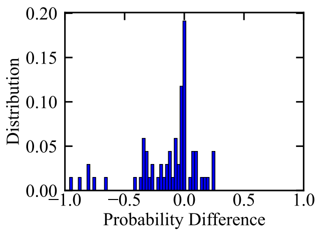
where Π represents the policy space; $\omega_k = \pi(\xi_k, \eta_k, \ldots, \xi_0, \eta_0)$; $\mathbb{E}_{\xi,\eta}$ denotes the expectation over the two stochastic processes of $\eta(t)$ and $\xi(t)$; the $A_k(\cdot)$ denotes the delay-mitigated mIoU in the k^{th} switching period. Note that the switching period may have multiple images, depending on the setting of the image frame rate. The objective is to find the optimal policy π^* that maximizes the average delay-mitigated mIoU per switching period.

Solving the policy optimization problem in Eq. (1) faces a basic challenge that the $A_k(\cdot)$ cannot be measured during ECSeg's online operations due to the unavailability of the image segmentation ground truths. Thus, in this study, we develop a learning-based controller to find the near-optimal solution. Specifically, during the offline training phase, the controller learns the optimal switching policy based on the real data traces including the image frames labeled with ground truths and cloud processing latencies collected from the ECSeg. Then, the learned policy is applied to determine the processing options during the online inference phase.

2) Markov property: When applied to image segmentation in AVs, ECSeg aims to switch between edge and cloud processing options based on image distance values and cloud transmission latencies. We conduct experiments to assess whether the above two stochastic processes satisfy the Markov assumption (MA), i.e., $\mathbb{P}[y_{k+1} \mid y_k] = \mathbb{P}[y_{k+1} \mid y_k, y_{k-1} \ldots y_0]$, where y_k represents the measurement state at the k^{th} time step. The MA suggests that the probability distribution of the state transition from y_k to y_{k+1} is independent of the past states $y_{k-1}, \ldots, y_0$. The MA is a basic property of the systems where RL is applicable [26]. When we assess the MA for the image segmentation applications, we use the probability difference, denoted by $\Delta P = \mathbb{P}[y_{k+1} \mid y_k] - \mathbb{P}[y_{k+1} \mid y_k, \ldots, y_{k-N}]$, where $N \geq 0$, as a MA compliance metric. A lower absolute value of ΔP indicates better compliance. Fig. 8 shows the distributions of ΔP with $N = 1$ for the transitions of the image distance values and the end-to-end cloud latencies in over 800 switching periods, where each switching period τ is one second. From Fig. 8, we can see that these two stochastic processes have good compliance with the MA because their values of ΔP concentrate at zero.



(a) Image distance.



(b) Cloud latency.

Fig. 8. Compliance of state transition with the Markov assumption.

B. MDP Formulation

System state: The system state, denoted by $\boldsymbol{x}$, is a vector $\boldsymbol{x} = [l, u]$, where l represents the average mobile network transmission latency in the last periods and the variable u represents image distance in the last period. The u is computed via the average pixel Euclidean distance of dense optical flow between the palette-based segmented results of the first frame and the last frame of the previous period. Computing u via the segmented results can avoid environmental impacts such as lighting, low texture, and contrast of adopting the dense optical flow methods on the raw image. A large value of l can decrease the delay-mitigated mIoU of the received cloud results. The u indicates the image distance. A larger image distance can result in a decrease in the delay-mitigated mIoU of the received cloud results.

Moreover, vehicle parameters, such as the ego vehicle's speed, may affect the delay-mitigated mIoU of the received cloud results. Intuitively, image distance between consecutive frames may increase with the ego vehicle's speed, leading to a larger mismatch between the received cloud results and the latest frame's ground truth. However, a small τ value, such as one second, can be set to monitor short-term image distances. In this case, the short-term historical image distance already incorporates the ego vehicle's speed. The ego vehicle's speed is redundant information and may introduce noise, misleading the switching controller.

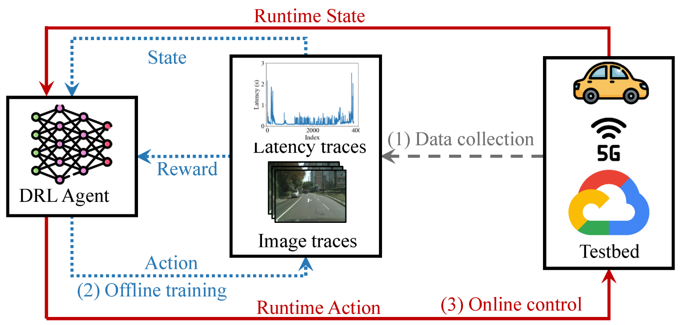


Fig. 9. Workflow of DRL-based switching.

Switching action: The switching action, denoted by $a \in \{0,1\}$, represents the choice between cloud and edge processing options to be executed in the current period.

Reward function: When an action a is performed at the current time step and the system state is $\boldsymbol{x}$, let $A_\tau(\boldsymbol{x}, a)$ denote the average delay-mitigated mIoU for image segmentation. $A_\tau(\boldsymbol{x}, a)$ varies in response to changes a and $\boldsymbol{x}$.

C. DRL Training

We adopt the learning framework of a DRL algorithm, called the proximal policy optimization (PPO) [27] to learn the optimal switching policy. PPO directly learns from a stochastic distribution, leading to more effective exploration, and adopts a clipping mechanism to improve efficiency compared with Deep Q-Network and Trust Region Policy Optimization. Under the typical setting, a PPO agent learns the optimal policy during the online interactions with the controlled system. However, for the formulated switching problem, the online DRL scheme faces the following two challenges. First, PPO agent trials may result in poor segmentation accuracy, potentially leading to safety concerns. Second, during the online learning phase, the mIoU cannot be measured during online learning due to the lack of ground-truth labels. To address these challenges, we adopt an offline training approach as illustrated in Fig. 9, which consists of three steps. First, we collect the real data traces from the deployment environment. Second, we use the real data traces collected in the first step to drive the offline training of the PPO agent. Third, the well-trained PPO agent is deployed to make decisions for switching.

VII. Evaluation

This section introduces the system settings and execution performance of ECSeg.

A. System Implementation and Experiment Settings

We conduct a series of experiments to investigate the performance of ECSeg in a real-world testbed. Fig. 10 illustrates the system setup. For image sensing, we use a See3Cam USB camera [28] with a 13-megapixel CMOS sensor, capturing images at 2048×1024 resolution at 17 Hz. We utilize a laptop equipped with an NVIDIA GeForce RTX 3080 Ti GPU as the edge platform mounted in the vehicle. For mobile network connectivity on the laptop, we utilize the SIM8202G module [29], which is a USB3.1 5G dongle equipped with four antennas and the Qualcomm Snapdragon X55 RF-modem. The cloud server is prototyped using a tower server with an RTX 8000 GPU and an Intel Xeon Gold 6246 CPU located in a university server room. We use Python 3.8's Multiprocessing library to manage concurrent processes, including image capturing, edge processing, and cloud processing. The InternImage model serves as the cloud model, using TensorRT 8.2.5.1. The deep learning-based segmentation model and PPO agents are implemented using PyTorch 1.11 and Tensorforce 0.6.5, respectively. The optical flow computing and image propagation are implemented using OpenCV 4.5.0.

We use the testbed to collect a self-collected real-world image dataset which comprises 12,036 road scene images captured with a USB camera at 17 Hz. The deadline is set to the image interval of 58.8 ms. During image capture, the testbed is deployed on a car that travels across various urban traffic conditions, including traffic jams, traffic lights, smooth traffic, and pedestrian crossings. The images are labeled with 19 classes. Moreover, we use the segmentation results obtained by executing the InternImage model as pseudo labels for the collected image. In addition to the self-collected dataset, we also use the Cityscapes dataset to evaluate our system.

We build a PPO-based DRL model with an input layer, three hidden layers and an output layer. The three hidden layers has 128, 64, and 32 ReLUs, respectively. The Adam optimizer with a learning rate of 10^{-3} is used for training. The PPO agent is trained with a likelihood ratio clipping set at 0.25 to balance stability during the learning process. Moreover, the switching period τ is set to 1 second. At the beginning of every period, the DRL agent observes a system state $\mathbf{x}$ including the image distance and cloud transmission latency. Then, it selects an action $a \in \{0, 1\}$ to choose the edge or cloud processing options. We employ ESPNet as the local model and InternImage as the cloud model. During the offline training phase, we utilize a trace-driven approach to accelerate the training process of the PPO agent. Specifically, the PPO agent makes decisions based on segmented image and latency traces. Subsequently, the edge and cloud model results' traces are utilized to compute intermediate reward values.

The offline training is conducted for 300 episodes, each of which includes 500 switching periods. In addition, we use 6,000 images from the Cityscapes dataset and 6,000 latency samples measured in a moving vehicle over six hours to train the PPO agent. Fig. 11 shows the PPO training traces of the rewards with various state inputs, including the latency and latency combined with image distance. Along the training episodes, the reward trace increases and then becomes flat under different inputs. The results show that the PPO agent can converge after a certain number of training episodes, and the agent with the state input combining latency and image distance achieves better convergence performance compared with single latency input.

B. Implementation Baselines

To assess the efficacy of the proposed ECSeg system, we compare our ECSeg with four baseline methods as follows:

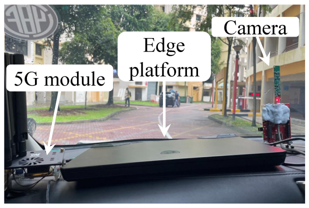


Fig. 10. Setup installed under a car's front windshield.

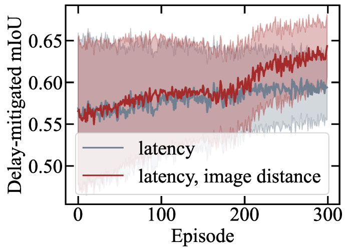


Fig. 11. Training results.

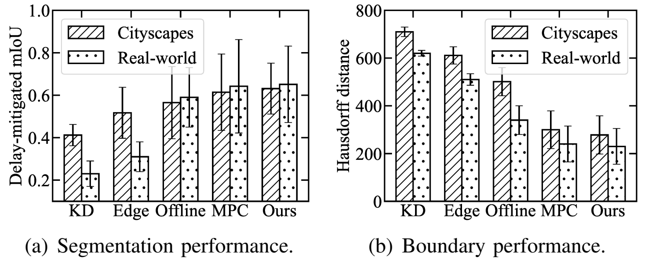


Fig. 12. Results from the Cityscapes and real-world dataset experiment.

Edge: The vehicle always executes the ESPNet [5] model to process all image frames without switching to the cloud processing option. *KD:* The vehicle always executes a lightweight model with five convolutional layers, trained using knowledge distillation with a soft-label-based loss function, to process all image frames without switching to the cloud processing option. *Offline:* Switching between edge and cloud processing options is based on thresholds for observed latency and image distance. These thresholds are determined using offline data traces [12]. *MPC:* Switching between edge and cloud processing options is determined by a model predictive control (MPC) method, similar to [13]. The MPC uses autoregressive moving average (ARMA) models to predict cloud transmission latencies and single-frame image distances, which are then used to compute the average image distance of received cloud results for the next switching period. An offline lookup table maps the average image distance to the delay-mitigated mIoU. To reduce computing overhead, the horizon length of MPC is set to one. Based on the image distance, an offline polynomial regression model estimates the delay-mitigated mIoU for the future period. The action with the highest delay-mitigated mIoU for the next switching period is selected.

C. Implementation Metrics

We adopt the following two metrics to evaluate our proposed ECSeg and baseline approaches. *Segmentation performance:* The average delay-mitigated mIoU is used as the metric to assess the accuracy of the image segmentation. *Boundary detection performance:* The image segmentation results can serve as input for higher-level applications, such as boundary detection in AVs. Boundary detection can help constrain AVs to avoid the collision, ensuring driving safety [30]. In this paper, we use boundary detection as a case study to investigate the performance of the ECSeg for high-level applications. In this case study, we use the metric of Hausdorff distance to measure the similarity between the all moving object boundary of the ground truth and the segmented results.

D. Execution Performance

1) Real-world experiments: Fig. 12(a) presents the delay-mitigated mIoU for the proposed ECSeg and four baseline approaches over a half-hour using the Cityscapes and self-collected datasets. The edge approach can achieve 0.505 delay-mitigated mIoU across all periods. The performance difference between the Cityscapes and real-world datasets occurs because the real-world dataset uses pseudo labels as ground truths for evaluation. These pseudo labels contain errors or misclassifications, leading to lower evaluation accuracy compared with using actual ground truths. Compared with the four baseline methods, our approach achieves the highest delay-mitigated mIoU and outperforms KD by over 48.8% in Cityscapes dataset. This is because the PPO agent makes good decisions to utilize accurate cloud results when the mobile network channel conditions are good. Our approach requires only 10 ms per decision, while the MPC method takes 400 ms, making it unsuitable for timely decisions within the deadline. Additionally, increasing the horizon length of the MPC leads to exponential growth in compute time.

Fig. 12(b) presents the moving boundary detection accuracy for the proposed ECSeg and four baseline approaches. The results show a similar trend to the image segmentation performance, with our approach achieving the lowest Hausdorff distance compared with the other four approaches. This is because the larger segmentation model deployed in the cloud provides accurate segmented pixels for detecting small moving objects. Moreover, our approach achieves a Hausdorff distance that is over 60.5% shorter than that of the edge approach.

2) Energy overhead: We further investigate the energy overhead of the mobile network communications and testbed. We measure the energy overhead by monitoring the laptop's battery level changes. Specifically, the energy overhead is computed as the difference between the initial and remaining battery levels over a 30-minute period without charging. Moreover, we also measure the mobile network module's energy overhead using the same approach, when it continuously transmits 500 KB of data to the cloud server for 30 minutes. The average mechanical system energy consumption of an electric car is about 0.15 kWh/km when driving in urban areas at an average speed of 30 km/h [31]. The mobile network module and the entire testbed consume only 0.0007 and 0.00267 kWh/km, respectively, which are less than 1% and 2% of the car's mechanical system energy consumption. These results show that the energy consumption of the testbed is negligible in the vehicle.

3) Evaluation in diverse cities: We conduct experiments to investigate the performance of ECSeg in different cities. We measure 20-minute cloud latency for one cloud server using the approach in §III-A, in two cities: City 1, close to the server, and City 2, over 1000 km away. In addition to the cloud geo-distance, the cities may have different mobile network data

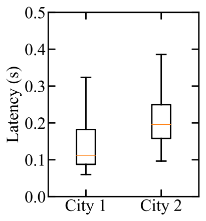


Fig. 13. Communication latency in two city.

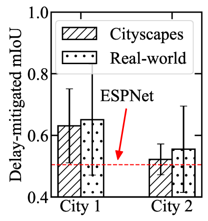


Fig. 14. ECSeg performance in two city.

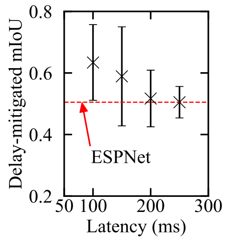


Fig. 15. ECSeg performance vs. latency.

transmission speeds, resulting in different latency. Note that the evaluation in §VII is conducted in City 1. Fig. 13 shows the box plots for the distributions of the data transmission latency in the two cities. The average latency in City 1 is about 110 ms, while in City 2 it is about 205 ms.

We evaluate ECSeg using these two latency traces on the Cityscapes and real-world datasets. Fig. 14 shows the average delay-mitigated mIoU for the two datasets under the two latency traces. Compared with City 1, the PPO agent's performance drops in City 2 for both datasets. For the Cityscapes dataset, the PPO agent's performance is only 0.517, close to the edge-based approach. This indicates that the PPO agent selects the edge processing option most of the time.

We further investigate ECSeg performance under different transmission latencies. Using the Cityscapes dataset and 10-minute latency traces from City 1 with various artificial biases, we simulate different latency situations. Fig. 15 shows ECSeg's mIoU under different average latency traces. The mIoU drops with average latency. When the average latency reaches 250 ms, ECSeg's mIoU is almost the same as the edge processing option because the PPO agent always selects the edge option over the cloud option. Therefore, cities prefer to have cloud servers within the city to support ECSeg.

VIII. Conclusion

This paper presents ECSeg, an edge-cloud switched image segmentation system for AVs applications. We formulate a switching problem aimed at maximizing the delay-mitigated mIoU of image segmentation under dynamic variations in mobile network conditions and image content changes. To address this, we apply deep reinforcement learning to learn the optimal switching policy for image segmentation processing. Our results demonstrate that cloud computing can benefit AVs, and ECSeg achieves a higher delay-mitigated mIoU for image segmentation in autonomous driving applications compared with four baseline approaches.

References

[1] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2017.

[2] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool, "Towards end-to-end lane detection: an instance segmentation approach," in *IEEE Intell. Veh. symposium*, 2018.

[3] J. Xu, Y. Nie, P. Wang, and A. M. López, "Training a binary weight object detector by knowledge transfer for autonomous driving," in *ICRA*, 2019, pp. 2379–2384.

[4] "Apollo." [Online]. Available: https://www.apollo.auto

[5] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi, "Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation," in *ECCV*, 2018.

[6] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6469–6486, 2020.

[7] X. Zhang, S. Ji, H. Wang, and T. Wang, "Private, yet practical, multiparty deep learning," in *ICDCS*, 2017.

[8] T. Kroeger, R. Timofte, D. Dai, and L. Van Gool, "Fast optical flow using dense inverse search," in *ECCV*, 2016.

[9] A. Hassan, A. Narayanan, A. Zhang, W. Ye, R. Zhu, S. Jin, J. Carpenter, Z. M. Mao, F. Qian, and Z.-L. Zhang, "Vivisecting mobility management in 5g cellular networks," in *SIGCOMM*, 2022.

[10] Y. Cai, T. Luan, H. Gao, H. Wang, L. Chen, Y. Li, M. A. Sotelo, and Z. Li, "Yolov4-5d: An effective and efficient object detector for autonomous driving," *IEEE Trans. Instrum. Meas.*, 2021.

[11] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *IEEE International Symposium Inf. Theory*, 2016, pp. 1451–1455.

[12] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *INFOCOM*, 2018.

[13] Z. J. Kong, Q. Xu, J. Meng, and Y. C. Hu, "Accumo: Accuracy-centric multitask offloading in edge-assisted mobile augmented reality," in *MobiCom*, 2023.

[14] X. He, S. Wang, X. Wang, S. Xu, and J. Ren, "Age-based scheduling for monitoring and control applications in mobile edge computing systems," in *INFOCOM*, 2022, pp. 1009–1018.

[15] T. Ren, Z. Hu, H. He, J. Niu, and X. Liu, "Feat: Towards fast environment-adaptive task offloading and power allocation in mec," in *INFOCOM*, 2023, pp. 1–10.

[16] S. Zhou, D. Van Le, R. Tan, J. Q. Yang, and D. Ho, "Configuration-adaptive wireless visual sensing system with deep reinforcement learning," *TMC*, 2022.

[17] "Orin." [Online]. Available: https://www.nvidia.com/en-sg/autonomous-machines/embedded-systems/jetson-orin/

[18] R. Poudel, S. Liwicki, and R. Cipolla, "Fast-scnn: Fast semantic segmentation network," in *British Mach. Vision Conference*, 2019.

[19] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *ECCV*, 2018.

[20] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *CVPR*, 2017.

[21] B. Cheng, I. Misra, A. Schwing, A. Kirillov, and R. Girdhar, "Masked-attention mask propagationer for universal image segmentation," in *CVPR*, 2022.

[22] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li *et al.*, "Internimage: Exploring large-scale vision foundation models with deformable convolutions," in *CVPR*, 2023.

[23] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *CVPR*, 2016.

[24] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. Qiao, "Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline," *NIPS*, vol. 35, pp. 6119–6132, 2022.

[25] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang *et al.*, "Planning-oriented autonomous driving," in *CVPR*, 2023, pp. 17 853–17 862.

[26] C. Shi, R. Wan, R. Song, W. Lu, and L. Leng, "Does the markov decision process fit the data: Testing for the markov property in sequential decision making," in *ICML*, 2020.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*, 2017.

[28] "See3cam camera." [Online]. Available: https://www.e-consystems.com

[29] "5g module." [Online]. Available: https://www.waveshare.com

[30] Z. Xu, Y. Sun, and M. Liu, "Topo-boundary: A benchmark dataset on topological road-boundary detection using aerial images for autonomous driving," *IEEE Trans. Robot. Autom.*, vol. 6, no. 4, pp. 7248–7255, 2021.

[31] W. Achariyaviriya, W. Wongsapai, K. Janpoom, T. Katongtung, Y. Mona, N. Tippayawong, and P. Suttakul, "Estimating energy consumption of battery electric vehicles using vehicle sensor data and machine learning approaches," *Energies*, vol. 16, no. 17, p. 6351, 2023.