SAGE: A Split-Architecture Methodology for Efficient End-to-End Autonomous Vehicle Control

ARNAV MALAWADE, MOHANAD ODEMA, SEBASTIEN LAJEUNESSE-DEGROOT, and MOHAMMAD ABDULLAH AL FARUQUE, University of California Irvine, USA

Autonomous vehicles (AV) are expected to revolutionize transportation and improve road safety significantly. However, these benefits do not come without cost; AVs require large Deep-Learning (DL) models and powerful hardware platforms to operate reliably in real-time, requiring between several hundred watts to one kilowatt of power. This power consumption can dramatically reduce vehicles' driving range and affect emissions. To address this problem, we propose SAGE: a methodology for selectively offloading the key energy-consuming modules of DL architectures to the cloud to optimize edge, energy usage while meeting real-time latency constraints. Furthermore, we leverage Head Network Distillation (HND) to introduce efficient *bottlenecks* within the DL architecture in order to minimize the network overhead costs of offloading with almost no degradation in the model's performance. We evaluate SAGE using an Nvidia Jetson TX2 and an industry-standard Nvidia Drive PX2 as the AV edge, devices and demonstrate that our offloading strategy is practical for a wide range of DL models and internet connection bandwidths on 3G, 4G LTE, and WiFi technologies. Compared to edge-only computation, SAGE reduces energy consumption by an average of **36.13%**, **47.07%**, and **55.66%** for an AV with one low-resolution camera, one high-resolution camera, and three high-resolution cameras, respectively. SAGE also reduces upload data size by up to **98.40%** compared to direct camera offloading.

CCS Concepts: • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Hardware** → **Power and energy**; • **Networks** → **Cyber-physical networks;**

Additional Key Words and Phrases: Energy optimization, edge computing, computation offloading, deep learning, autonomous vehicles

ACM Reference format:
Arnav Malawade, Mohanad Odema, Sebastien Lajeunesse-DeGroot, and Mohammad Abdullah Al Faruque. 2021. SAGE: A Split-Architecture Methodology for Efficient End-to-End Autonomous Vehicle Control. *ACM Trans. Embedd. Comput. Syst.* 20, 5s, Article 75 (September 2021), 22 pages.
https://doi.org/10.1145/3477006

A. Malawade and M. Odema contributed equally to this research.

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2021.

This work was partially supported by the National Science Foundation (NSF) under award CMMI-1739503 and Graduate Assistance in Areas of National Need (GAANN) under award P200A180052. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agency.

Authors' addresses: A. Malawade, M. Odema, S. Lajeunesse-DeGroot, and M. A. Al Faruque, University of California Irvine, Irvine, USA; emails: {malawada, modema, slajeune, alfaruqu}@uci.edu.

1539-9087/2021/09-ART75
https://doi.org/10.1145/3477006

1 INTRODUCTION AND RELATED WORK

Advances in deep learning, hardware design, and modeling over the past decade have enabled the dream of autonomous vehicles (AVs) to become a reality. AVs are expected to improve road safety, passenger comfort, and mobility significantly. The core task of an AV is to perceive the state of the road and safely control the vehicle in place of a human driver. The difficulty in achieving this goal lies in the fact that road scenarios can be highly complex and dynamic, presenting a wide range of potential challenges and obstacles (e.g., rain, snow, construction zones, animals, etc.). To address this challenge, AV algorithms rely heavily on (i) large deep learning (DL) models to capture this high degree of complexity and (ii) high-performance edge, hardware to reduce processing latency and ensure passenger safety at higher speeds.

As a result, AVs require significant computational power to operate reliably and safely in the real world. However, as AV computing capabilities have scaled up, so have their power and energy requirements. For example, the Nvidia Drive PX2, used in 2016-2018 Tesla models for their Autopilot system [1], can achieve 12 Tera Operations Per Second (TOPS) with a Thermal Design Power (TDP) of 250 Watts (W). Following the PX2 was the Nvidia AGX Pegasus, which was built for level 5 autonomy; it can achieve 320 TOPS with a TDP of 500 W [33]. What's more, the next generation hardware platform using the Nvidia AGX Orin SoC is expected to be capable of 2000 TOPS with a TDP of 800 W [2]. Although AV hardware platforms are becoming more efficient in terms of TOPS/W, the baseline energy demands continue to increase as more advanced DL models and hardware platforms are developed. The increased power demands of these systems also increase the heating, ventilation, and air conditioning (HVAC) system's thermal load. The combined computational and thermal loads of these platforms can reduce an AV's driving range by up to 11.5% [28], which is especially detrimental for electric vehicles due to their limited range and long recharge times.

Researchers attempting to address this problem for AVs as well as other cyber-physical systems have proposed several approaches for reducing energy consumption, including application-specific hardware design, cloud/fog server offloading, or model simplification/pruning [3, 26, 28, 32, 35, 39]. Although solutions like Application-Specific Integrated Circuits (ASICs) can reduce energy consumption through hardware optimization, they are prohibitively expensive to develop. Furthermore, with ASIC designs, all model specifications and contingencies need to be accounted for at design time, meaning there is little to no support for adding new features, fixing algorithmic errors, or modifying model architectures. Costly development stages will need to be repeated for every revision to the model. The next logical choice is to attempt model simplification/pruning without changing hardware platforms; however, it is difficult to significantly reduce energy consumption by pruning without adversely affecting the AV's performance and safety. To address the limitations of the previous two approaches, some works propose offloading some or all AV tasks to the cloud for processing to reduce the energy consumption of the AV without changing the hardware or algorithms. Unfortunately, current offloading approaches have significant scalability and latency issues, as will be discussed in the next paragraph. In contrast, we propose a cloud server offloading methodology that is efficient, safe, and practical for current networking infrastructure.

A naïve solution to the problem of edge, energy consumption is to offload self-driving tasks to a cloud server or a Mobile Edge Computing (MEC) server [13]. These 'direct offloading' approaches involve sending images or sensor inputs directly to the server, which processes the data before returning the desired control outputs to the vehicle. However, the real-time latency constraints of autonomous driving and the limitations of current wireless network infrastructure significantly impact this solution's feasibility; to drive and react effectively, AVs must be able to process each input within 100 milliseconds [28]. This bound comes from the fact that the fastest attainable reaction by a human when driving falls within the range of 100–150 ms, meaning that for efficient AV navigation, AVs need to at least perform at the same level as the human driver counterpart. Additionally, most real-world AVs, such as those from Tesla [1], Baidu Apollo [20], and Argo AI [21], use multiple high-definition cameras and sensors and would require very high network bandwidths to offload data within the latency constraints. In some cases, the energy needed to transmit and receive data from the cloud server can even *exceed* the energy consumed by edge-only processing. Together, these factors make direct offloading infeasible in most real-world autonomous driving scenarios. Currently, most of the literature has proposed solving this problem by improving network robustness and throughput via solutions such as 5G C-V2X [34] and WAVE [12], or even by placing sensors on the roads themselves [26]. However, implementing these solutions would require significant investments in the networking infrastructure to become realistically feasible.

Several works have proposed methods for offloading some or all AV tasks. For example, [10] proposed a technique for reducing AV processing latency by offloading sub-tasks of LiDAR SLAM to the cloud depending on network conditions. Although they demonstrate good performance, their approach is limited since it only considers LiDAR data, which is significantly smaller than camera data. Additionally, they developed a distributed SLAM algorithm that allowed task-level parallelism; this sort of optimization will need to be applied for every part of a modular AV pipeline and may not be applicable in some areas. In another work, [36] proposed an offloading strategy where computations are executed on either an MEC server or a cloud server depending on network conditions. However, their method requires all sensor input data and internal state information to be sent to the server for processing. Since they only evaluated a micro-car transmitting IMU data (position, velocity, yaw), their approach is not scalable to real-world AVs that would need to offload multiple high-definition camera inputs. The work in [41] proposes a hierarchical approach for offloading in which AVs can offload to road-side units (RSUs) when MEC servers are overloaded, but this work does not consider network bandwidth constraints. Moreover, none of these works [10, 36, 41] considered edge, energy consumption in their evaluation, which significantly constrains direct offloading approaches. The authors in [42] evaluated the energy consumption for offloading to MEC servers; however, they do not assess this approach's practicality for large upload data sizes, which are typical for AVs with multiple high-resolution input cameras. In summary, the problem of offloading large data sizes while meeting latency and energy constraints on current network infrastructure is exceedingly challenging and is currently unsolved by existing methods.

1.1 Research Challenges

For efficient AV offloading, the following key research challenges need to be addressed:

(1) Offloading AV tasks without exceeding safety-critical latency constraints or increasing AV energy consumption.
(2) Adapting AV deep learning architectures to support dynamic offloading depending on the corresponding network conditions.
(3) Developing a technique efficient enough to meet latency constraints with data from multiple high-definition camera inputs on current industry-standard AV hardware.
(4) Producing a cost-efficient, safe solution that can operate within the constraints of current networking infrastructure.

Instead of altering the AV hardware or the communication network infrastructure, we propose SAGE: a methodology to significantly reduce the size of the data transmitted over the network and enable efficient computation offloading. By introducing a *bottleneck* layer near the beginning of end-to-end DL control models, the size of the data uploaded to the cloud server is reduced significantly, allowing a large portion of the model computation to be offloaded to the server even at

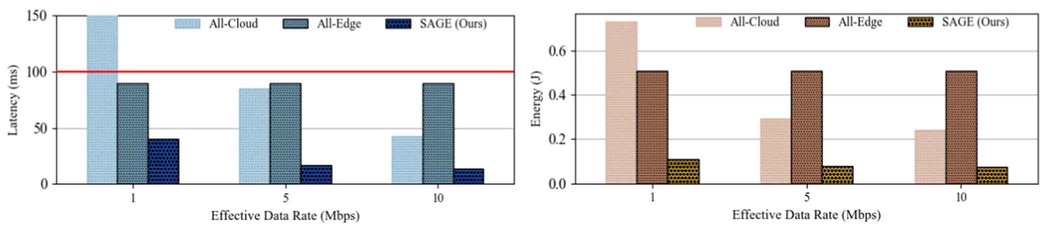

Fig. 1. A comparative analysis demonstrating the of the overall latency *(left)* and energy consumption *(right)* for the *All-Cloud*, *All-Edge*, and *SAGE (Ours)* execution strategies given three typical effective data rate values attainable through a 4G LTE connection. The red line indicates the AV processing latency deadline of 100 ms.

low network bandwidths. This benefit is especially valuable in multi-camera offloading due to the significant bandwidth requirements and edge, energy consumption of multi-camera models. Furthermore, it was shown in [30] that, with a particular training strategy, the model's performance after introducing the *bottleneck* remains nearly the same.

1.2 Motivational Example

We provide a brief example to demonstrate the merit of our approach in Figure 1. Here, we compare three possible execution strategies for an end-to-end AV control model: executing locally on the edge, (*All-Edge*), offloading the entirety of execution to the cloud (*All-Cloud*), and our proposed split approach (SAGE). Our analysis is conducted at three distinct data rate values typical for 4G LTE connections, and the evaluations are performed on a Jetson TX2 for a ResNet-50 model [15] adapted for end-to-end AV control. In terms of latency, it is clear that the *All-Cloud* approach is impractical at low data rate values as it fails to meet the 100 ms processing latency constraint of the AV. On the other hand, performing all the processing locally in the *All-Edge* approach keeps the latency unaffected by the state of the network. However, the downside is that the edge, device is fully operational and consumes sizeable amounts of power for longer periods, leading to higher energy consumption in theory than the *All-Cloud* approach at the more favorable data rates. SAGE offers to leverage the best of these both approaches. In brief, SAGE entails replacing an early computational block from the model architecture with a more efficient encoder-decoder-like structure. Then, this modified architecture is divided between the edge, and cloud at the encoder output. The encoder, acting as a *bottleneck*, projects the input data into a low-dimensional representation that is more suited to be transmitted to the cloud over the wireless medium. On the other hand, the decoder component is situated as part of the cloud to receive the encoder's output data and map it into a representation analogous to the output of the original computational block from the unaltered model architecture. This structural modification results in significantly lower: (i) local execution latency than the *All-Edge*, and (ii) transmission latency than the *All-Cloud*. Moreover, these improvements are reflected in the energy consumption as the edge, device is only required to perform computations for a much shorter interval within the 100 ms time window, which is beneficial for the edge, device itself in terms of performance efficiency. More details about the proposed SAGE methodology and how resource-efficiency is promoted across the edge, and cloud while maintaining the same degree of accuracy shall be described in detail in Section 3.

1.3 Novel Contributions

Our paper presents the following contributions.

(1) We propose a novel split-network architecture methodology that allows for a significant reduction in the energy consumption of AV on-board processing units by dynamically offloading part of the model's computations to the cloud.
(2) We demonstrate that introducing *bottlenecks* into deep end-to-end AV control models reduces energy consumption significantly with little to no performance loss.
(3) We show that SAGE reduces network throughput requirements significantly compared to conventional cloud server offloading techniques, enabling it to meet latency constraints even at low network bandwidths on 3G, 4G LTE, and WiFi.[1]
(4) We demonstrate that SAGE is scalable to practical AV use cases by evaluating its performance with three high-definition camera inputs, typical for real-world AVs [1, 16, 20, 21].
(5) We demonstrate the practicality and feasibility of our technique by evaluating its performance on the Nvidia Jetson TX2, as well as the industry-standard Nvidia Drive PX 2 autonomous driving platform, used in all 2016-2018 Tesla models for their Autopilot system [1].

1.4 Paper Organization

The remainder of the paper is organized as follows. In Section 2, we discuss our system model and problem formulation. In Section 3 we elaborate on SAGE's design methodology. In Sections 4 and 5 we present and discuss our experimental results. Finally, in Section 6 we present our conclusions.

2 SYSTEM MODEL

This section aims to provide a generalized model of how an AV edge, device may complete processing a task either through local computation or collaboration with a cloud server. Mainly, the modeling comprises the communication and computation costs that the AV edge, device would incur until the task is finished. Our model comprises a direct link between a vehicle i, requiring computation for its designated task, and a cloud server j to whom tasks can be offloaded.

2.1 Communication Model

As the AV runtime optimization solution spans multiple levels in the system architectural hierarchy (i.e., edge, and cloud), a communication model is needed to identify the cost of transferring data between entities of different levels. These costs can be represented through transmission latency and energy. More formally, the task to be offloaded can be represented as $t_i = \{a_i, b_i, c_i\}$, where a_i, b_i, and c_i correspond to the size of data to be transmitted, size of data to be received back from the server, and the number of CPU cycles required to complete the task, respectively. To estimate the communication overhead, we will need to determine the upload and download data rates, $r^U_{i,j}$ and $r^D_{i,j}$, experienced at vehicle i's edge, device when transmitting data to cloud server j. Although the data rate can be determined theoretically through Shannon's law, this resembles an optimistic estimate, not taking into consideration potential errors or packet losses. Instead, we are more interested in the '*effective*' data rates by which we mean the actual data transfer speeds experienced at the edge, device when accounting for errors and re-transmissions. These values can be measured at the target device and accordingly, the upload and download latencies can be given as:

$$T^U_{i,j} = \frac{a_i}{r^U_{i,j}}, \qquad T^D_{i,j} = \frac{b_i}{r^D_{i,j}} \tag{1}$$

[1]We did not evaluate 5G C-V2X and WAVE in this work because these technologies are currently not widespread and require significant infrastructure changes to be viable. Also, comparable real-world power models for 5G are not available yet in the literature. However, SAGE will be scalable to these emerging technologies.

Thus, the total communication overhead encountered by at vehicle i in terms of latency and energy for offloading task execution to computing server j is given by:

$$T_{i,j}^{comm} = T_{i,j}^{U} + T_{i,j}^{D} + T_{i,j}^{RTT} \tag{2}$$

$$E_{i,j}^{comm} = p_i^T T_i^U + p_i^R T_i^D \tag{3}$$

where p_i^T, p_i^R and $T_{i,j}^{RTT}$ represent vehicle i's transmitting power, receiving power, and the round-trip time between vehicle i and server j, respectively.

2.2 Computation Model

Assuming that any task requested by vehicle i consists of several sequential sub-tasks, i.e., as in an end-to-end control pipeline or layers in a DL model, let $\mathbb{C}_i = \{c_{i1}, c_{i2}, \ldots, c_{iK}\}$ denote the set of K clock cycles required to execute each sub-task. Thus, potential execution times (local or remote) and the energy needed to execute sub-task k locally are:

$$T_{ik}^{l} = \frac{c_{ik}}{f_i^l} \tag{4}$$

$$T_{ik}^{r} = \frac{c_{ik}}{f_i^r} \tag{5}$$

$$E_{ik}^{l} = \vartheta_i c_{ik} \tag{6}$$

where f_i^l, f_i^r and ϑ_i represent the operational frequency at vehicle i, operational frequency at the remote server, and a coefficient denoting energy consumed per CPU cycle at vehicle i. However, since offloading some or all sub-tasks is a viable option in this scheme, the total computational latency and energy consumption for vehicle i can be written as:

$$T_i^{comp} = \sum_{k=1}^{k_p} T_{ik}^{l} + \sum_{k=k_p+1}^{K} T_{ik}^{r} \tag{7}$$

$$E_i^{comp} = \sum_{k=1}^{k_p} E_{ik}^{l} + E_i^{idle}(t) \tag{8}$$

where k_p is the execution partitioning point after which execution is assigned to the remote server, and $E_i^{idle}(t)$ is the energy consumed by vehicle i waiting for the remote server's results as a function of the idle time t. Note that when $k_p = K$, T_i^{comp} reflects the local execution case without any form of offloading as the second summation becomes an empty sum.

2.3 Problem Formulation

From the previous model derivations, the offloading problem for vehicle i can be formulated as:

$$\min_{k_p} w_i^T \left(\mathcal{I}(k_p \neq K) \times T_i^{comm} + T_i^{comp}\right) + w_i^E \left(\mathcal{I}(k_p \neq K) \times E_i^{comm} + E_i^{comp}\right) \tag{9}$$
$$\text{s.t. } \left(\mathcal{I}(k_p \neq K) \times T_i^{comm} + T_i^{comp}\right) <= 100\ ms$$

where w_i^T and $w_i^E \in [0, 1]$ represent user-defined weights associated with the latency and energy metrics, and $\mathcal{I}(k_p \neq K)$ is an indicator function becoming 0 in the case of local execution. As presented earlier, the 100 ms constraint is the window within which the AV must finish its processing task [28]. Note that as k_p varies, so will the values associated with the offloaded task a_i and c_i.

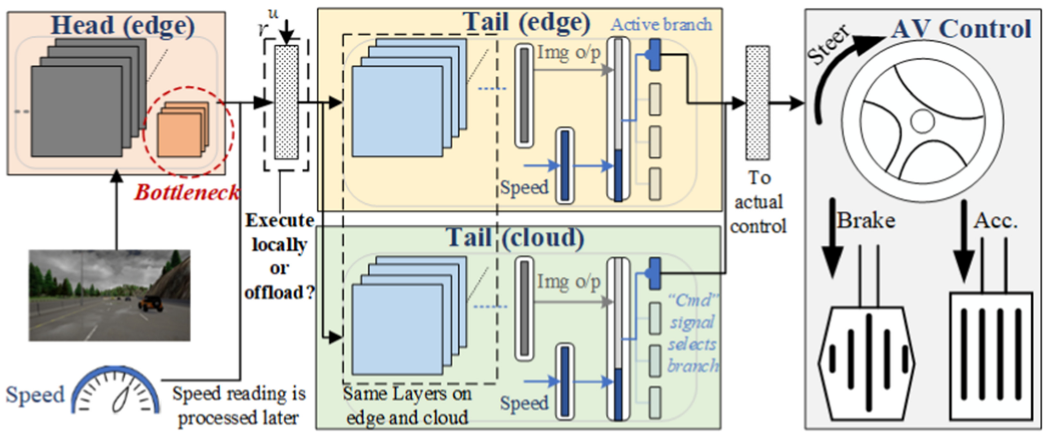

Fig. 2. An illustration of how systems developed through SAGE support end-to-end AV control. The tail component is replicated on both the edge, and the cloud. At runtime, a decision is to be made whether the tail should be executed locally or in the cloud. Final results are applied as inputs to the AV control system.

3 SAGE DESIGN METHODOLOGY

In this section, we present SAGE and discuss its building blocks in detail. Figure 2 illustrates how the final system developed through SAGE would support end-to-end AV control. The implemented DL model is divided into two components: (i) a *head* deployed on the edge, and (ii) a *tail* which is replicated across both the edge, and cloud. The *head* component contains within its structure a *bottleneck* layer, which represents an optimal offloading point compared to other options. Inputs to the model can come through a camera feed and sensory measurements (e.g., current speed). After the head portion executes at runtime, it is decided whether tail processing should be done locally or be delegated to the cloud depending on current network conditions. The *tail* portion of each DL model contains the bulk of layers and outputs the control values to the AV.

3.1 Perception

Much like human beings, perception is concerned with how an AV interprets and understands events occurring in its surrounding environment. To enable perception, AVs are equipped with sensory capabilities to capture representative data from the environment. This data is then processed to extract a comprehensive understanding of the events unfolding around them. Contemporary AVs sense their environment via cameras, LiDAR, or radar equipment [14, 25, 27]. After data acquisition, DL models process the data and estimate the course of action that the AV should take in the following time-step [8]. Without any loss of generality, our evaluations are based on the data-intensive image-based perception from a set of cameras capturing the AV's surroundings. To implement the perception pipeline, we utilize state-of-the-art DL model architectures, which are known to achieve high accuracy on image classification tasks, as baselines. This allows us to leverage these models' abilities to capture fine-grained features from images for processing camera data as part of an end-to-end AV control architecture. Mainly, we consider DenseNet-169 [18], ResNet-34 [15] (used for end-to-end multi-camera AV control in [16]), ResNet-50 [15], and CarlaNet [8] which is implemented specifically as an end-to-end AV control solution.

3.2 Imitation Learning for End-to-End Autonomous Vehicle Control

Next, the baseline models must be adapted to predict AV control outputs from camera input data. This can be achieved by integrating an Imitation Learning (IL) component at the back-ends of the

Table 1. Contribution of Perception and IL Components in Terms of the Total Processing *(top)*, and How Modifying the *head* Components in SAGE Speeds up Model Executions *(bottom)*

	DenseNet-169 [18]	ResNet-50 [15]	ResNet-34 [15]	CarlaNet [8]
Perception	98.76%	97.74%	97.36%	59.64%
Imitation Learning (IL)	1.24%	2.26%	2.64%	40.36%
Modified *head* speedup	80.11%	79.97%	67.25%	13.39%
Overall Model speedup	27.01%	41.51%	34.39%	2.65%

baseline models to enable them to mimic a human's behavior in regard to a particular task. In this context, the driving algorithm's core objective is to *imitate* the vehicle control outputs (steering angle, brake pedal angle, and accelerator pedal angle) produced by a human driver for a given set of input images [37]. IL models are typically trained via supervised learning, where the goal is to map the input features captured at time-step t to the corresponding human control output values. To effectuate the learning process, a loss function, e.g., Mean Absolute Error (MAE), is used to evaluate the difference between a model's predictions and the ground truth values. Take the baseline ResNet-50 for example, its vanilla network architecture constitutes five main convolutional blocks, representing the main perception component, followed by a final fully-connected layer for image classification tasks. To adapt the model for IL, we replace this fully-connected layer with an IL component developed for end-to-end AV control where the final layer has three separate neurons: one for each control output (steering, accelerator, brake). These outputs are used in both the loss function for MAE computation and controlling the vehicle during deployment. We follow the IL implementation in [8] where firstly, the output from the preceding perception component and the corresponding pre-processed speed measurement at time-step t are concatenated together as the input to the IL component. Next, one of several processing branches is activated based on the driver's command value (e.g., navigation signal). This notion of branching is implemented to associate unique learning features with different driving intentions. For instance, the second branch can only be activated whenever the driver issues the *"Turn right"* navigation signal because this branch's parameters were trained to take actions in anticipation of a right turn, dissimilar to what parameters in other branches learned. The outputs from the active branch are the ones that are directly applied to the AV control system at that particular time step t.

To summarize, a baseline DL-based solution for AV control comprises (i) a perception module, (ii) a speed measurement processing unit, and (iii) an IL back-end. Henceforth, these DL models adapted for IL shall have *"IL-"* preceding their original names, e.g., *IL-ResNet-50*. Moreover, to give an idea, of each component's contribution to the overall processing time, The upper part of Table 1 shows how perception can be the most computationally-intensive component, especially when utilizing state-of-the-art image classification models. Note that the speed processing unit is executed concurrently with the perception module, which dominates their combined execution time. Thus, our structural modifications target the perception modules to maximize the performance impact.

3.3 Structural Alterations for Split Computing

Deploying the AV control algorithm on the edge, device is essential for such a mission-critical application. However, dynamically assigning some or all of the processing tasks to a more powerful cloud server if the wireless network conditions are favorable can lead to substantial latency and energy savings on the edge, device. As was shown in the motivational example, directly offloading inputs to the cloud can be inefficient at sub-par network conditions: a significant communication overhead can arise from transmitting the raw input images, resulting in a poorer overall performance than that of local execution. Prior work in [40] tries to address this by compressing the input image before transmission, but accuracy degrades significantly. One alternative in [23] proposes scanning each layer within the DL model to identify those which output smaller data sizes than the input as potential data offloading points in a split computing approach. However, this is dependent on each architecture's structure, deeming it ineffective for models that do not shrink data size enough.

A more tractable alternative is modifying the DL model structure by injecting a *bottleneck* amongst the first few layers. This *bottleneck* layer presents an optimal offloading point very early in the model because its structure is designed to output exceedingly small-sized data. This idea, is presented in [29–31], where it is implemented by initially dividing a DL model architecture into two sections: a *head* and a *tail*. The structure of the *tail* remains unchanged. Whereas, a simpler more efficient version of the *head* is constructed to mimic the functionality of the original *head* section. The merits of constructing this new *head* model are twofold. Firstly, the new *head* is structurally more efficient to run than the original *head* providing a local execution speedup, as illustrated in the lower part of Table 1. Secondly, the *head* contains the *bottleneck* operating as an encoder-decoder model rigorously transforming its input to lower dimensions (encoder) before raising the dimensionality at its output (decoder), making the encoder serviceable as an efficient data offloading layer. We follow the instructions provided in [30] on how to design a new *head* model with a *bottleneck* from the original *head*. Structurally, both the *bottleneck*'s number of output channels and its preceding layers' complexity should be minimized. However, the modified model's accuracy still needs to be maintained by retraining the new head portion, as discussed in the following subsection. The overhead for creating a bottleneck layer is analogous to that of creating a small deep learning model manually, which is represented through the human design effort of performing successive refinements in order to attain the desired degree of performance. The main difference is the requirement to have an encoder-decoder structure within the overall architecture to provide the efficient offloading point.

As an example to demonstrate the efficacy of the *bottleneck*, we compare performances at an injected *bottleneck* in DenseNet-169 [18] against offloading at the input or one of the six layers that provided the *smallest* output (o/p) sizes in the original DenseNet-169. In this analysis, shown in Figure 3, we assume a 10 Mbps connection using the 200×88 sized images from [8] to calculate the overall latency with the Jetson TX2 as the edge, device. Layers other than the *bottleneck* are displayed according to their position within the DenseNet-169 architecture. The following trend can be observed from Figure 3: as we go deeper in the network, the size of the output data at each layer decreases, reducing transmission overheads as we progress. However, to reach those lower layers, a considerable amount of execution needs to be performed locally. Thus, none of these deployment options outperform offloading at the input layer. On the contrary, injecting a *bottleneck* early in the architecture decreases the output size to a value much smaller than the input, reducing transmission overhead. Moreover as a result of its early placement, intensive local processing is not required prior to the *bottleneck* layer. All in all, offloading at the *bottleneck* is 14× faster than offloading at the input. This result would also be reflected in the energy consumption. From a formal perspective, the *bottleneck* dominating all other offloading strategies transforms Equation (9) into a runtime binary decision problem, in which either local execution is selected at extremely poor network conditions or offloading at the *bottleneck* is chosen otherwise.

3.4 Head Network Distillation (HND)

Knowledge Distillation (KD), presented in [5, 17], has emerged as an effective training technique to render a compressed yet accurate version of a deeper, more complex neural network model. The main reason this technique came about is that shallower neural networks, when trained conventionally, achieve sub-par performance at many tasks compared to deeper networks. Hence, this

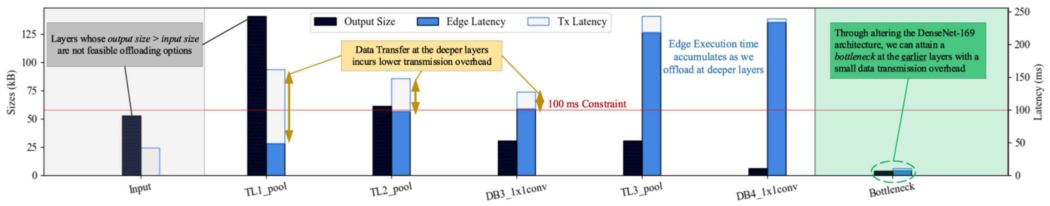

Fig. 3. A comparison between offloading from: (i) the input, (ii) different perception layers in IL-DenseNet-169 [18], or (iii) the proposed *bottleneck* layer. The edge, device is a Jetson TX2, the input is a 200 × 88 sized image, and the effective data rate is 10 Mbps. The IL-DenseNet-169 layers displayed are the ones which provided the smallest output data size. Note that the only options that do not violate the 100 ms latency constraint are offloading at the input or offloading at the *bottleneck*, with the latter offering a 14× overall speedup.

technique aims to leverage the deeper network as a *teacher* to *distill* its acquired knowledge into the smaller *student* model. Consequently, *student* models trained through KD achieve superior performance relative to their traditionally-trained counterparts [5]. This technique is advantageous when high-performance neural network solutions are needed for edge, devices with limited resources. Formally, the *student's* loss function, which is minimized during training, needs to incorporate a distillation component as follows:

$$\mathcal{L}_{student} = \alpha \mathcal{L}_{orig} + (1 - \alpha)\mathcal{L}_{KD} \tag{10}$$

where $\mathcal{L}_{orig}$ is the conventional loss function using hard labels, whereas $\mathcal{L}_{KD}$ represents the KD loss component, which can be computed using KL divergence, L2 loss, or logits regression [5]. Through providing a control variable α, the effective weight of each loss component can be fine-tuned. This works because the *student* is learning by minimizing the divergence from a vector of the *teacher's* real values, rather than on a single label representation. Hence, the student becomes more capable of capturing the finer details of how the final decision was reached and attempts to learn a simple function to minimize the divergence from this vector of values, thus achieving better generalization.

However, works in [6, 38] discuss how using more complex and accurate *teacher* models makes training through KD for the *student* models more challenging as a result of the capacity mismatch. In these scenarios, more training heuristics are introduced, and more restrictions are imposed on the structures of *student* models. To avoid this in the context of the AV problem, KD is applied between the original and modified *head* components rather than the entirety of models, making them the *teacher* and *student*, respectively. This process entails training the learnable parameters within the modified head model while maintaining the pre-trained tail parameters unchanged from the original model. Consequently, the loss component for the *student-head* model can be computed using the sum of squared difference, as presented in [30]:

$$\mathcal{L}_{KD}(X) = \sum_{x \in X} ||s_h(x) - t_h(x)||^2 \tag{11}$$

where s_h and t_h represent the output vectors from the head portions of the *student* and *teacher* on an input x, respectively. For this loss function to be attainable, the final layers in both *head* models must have the same dimensions. Figure 4 illustrates the Head Network Distillation (HND) process. Note that although the loss function is computed between the final layers in the *head* modules, in the final deployment, any layers succeeding the *bottleneck* are deployed on the cloud.

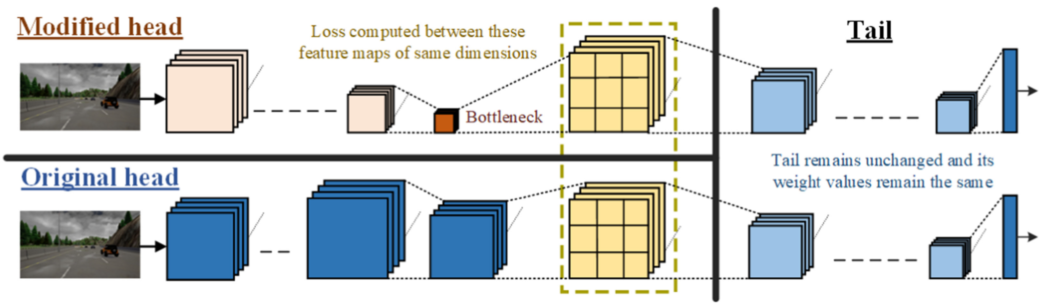
Modified head
Loss computed between these feature maps of same dimensions
Bottleneck
Tail
Original head
Tail remains unchanged and its weight values remain the same

Fig. 4. The head network distillation process to train the perception component with the *bottleneck*. Note that in the final deployment, the *bottleneck* layer is to be the last layer on the edge, device.

3.5 Offloading Strategy

After training the modified model using HND and deploying it across the edge, and cloud, a run-time strategy must be implemented to determine, for each time step, whether to continue execution at the *bottleneck* or delegate the remaining DL processing tasks to the cloud. The corresponding network conditions, mainly the effective upload data rate: $r^U_{i,j}$, govern this decision. Note that the focus is on $r^U_{i,j}$ because the bulk of the data transmission (tens of kBs) occurs in the uplink, whereas merely the final values (in bytes) are sent through the downlink. So, the task here is to devise a policy based on a data rate threshold r_{th} where:

(1) if ($r^U_{i,j} > r_{th}$): the edge, device offloads the result of computation at the *bottleneck* to the cloud server where it is processed through the *tail* part of the model before sending the resultant control inputs back to the edge, device.
(2) *else:* execution proceeds locally at the edge, device.

To estimate r_{th}, the 100 ms constraint on AV processing, stated in Equation (9), must be considered, where all communication- and computation-related tasks must conclude within that time frame. Moreover, there have to be expected performance gains to justify the offloading decision. Therefore in our formulation, this decision is dependent on whether or not there exist potential energy reductions from offloading. Hence, we can denote r_{th} as:

$$r_{th} = \frac{Upload\ Data\ Size}{100 - \left(T^{edge}_{head} + T^{cloud}_{tail} + T^D_{i,j} + T^{RTT}_{i,j}\right)} \quad s.t. \;\; (r_{th} > 0)\ and\ \left(E^{comm}_i + E^{edge}_{idle} < E^{edge}_{tail}\right) \qquad (12)$$

where T^{edge}_{head} and $T^{RTT}_{i,j}$ represent the edge, *head* component's execution time and the round-trip time between the edge, and cloud, respectively. The sum of T^{cloud}_{tail} and T^D represents the time the edge, device is idle waiting for the cloud server to compute and transmit back the control inputs for the AV. The $r_{th} > 0$ restriction guarantees that the sum of the latency estimates in the denominator does not exceed the 100 ms time constraint. Furthermore, the sum of the energy required to offload the data at the *bottleneck* and the idle energy consumption ($E^{comm}_i + E^{edge}_{idle}$) must be less than the energy required to execute the tail component of the model (E^{edge}_{tail}) in order to attain a *beneficial* offload. Algorithm 1 demonstrates a runtime algorithm implementing this strategy. We have built this algorithm to promote performance efficiency through offloading whenever the network conditions are benign. Note that ***lines 12–14*** represent a fail-safe mechanism accounting for the network variability *within* a single time window, where it is vital to keep room within the 100 ms time window to invoke local tail execution if the result has not received from the cloud

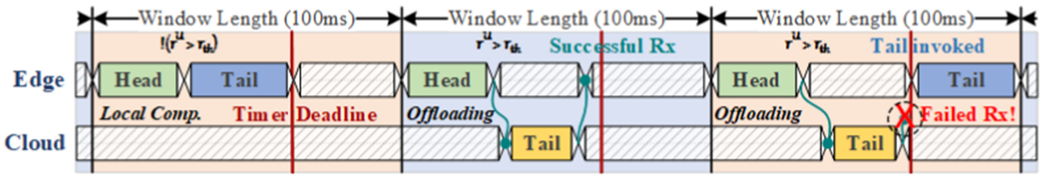

Fig. 5. Three possible execution scenarios: local execution, successful offloading to the cloud, and unsuccessful offloading to the cloud which entails rolling back to edge, computing.

ALGORITHM 1: Runtime Energy Optimization Algorithm

```
Input: Upload Data Size, $T^{edge}_{head}$, $T^{cloud}_{tail}$, $E^{edge}_{idle}$, $E^{edge}_{tail}$
for each time step t do
    Measure $r^U_{i,j}$, $r^D_{i,j}$, and $T^{RTT}_{i,j}$        // obtain current network parameters
    Calculate $T^D_{i,j}$, $r_{th}$, and $E^{comm}_i$        // using current network parameters
    $x = edge\_head()$        // execute locally until bottleneck
    if $(r^U > r_{th})$ and $(r_{th} > 0)$ and $(E^{comm}_i + E^{edge}_{idle} < E^{edge}_{tail})$ then
        $Tx\_data(x)$        // transmit bottleneck output to the cloud server
        $Timer \leftarrow reset()$        // initialize timer
        $edge\_state \leftarrow idle$        // edge goes to idle mode
        if $rx\_event$ then
            $edge\_state \leftarrow wakeup$        // edge wakes up to receive server results
            $x = Rx\_data()$
        else if $Timer > (100 - (T^{edge}_{tail} + \epsilon))$ then
            $edge\_state \leftarrow wakeup$        // edge wakes up to execute tail model
            $x = edge\_tail(x)$
    else
        $x = edge\_tail(x)$        // execute tail model locally
    $Input\_Control(x)$        // apply control values to the AV
```

within the expected time limit. This is guaranteed by starting a counter each time window that wakes the edge, device to resume execution if the remaining time within the window is equivalent to that of the edge, tail model. Also, $T^{RTT}_{i,j}$ in our case is obtained through averaging multiple pings to a remote server, which accounted for $< 10ms$ overhead, however, this value may vary depending on the operational scenarios and the capabilities of network components involved in the communication link. Figure 5 illustrates the three possible outcomes from our runtime strategy. It should be noted that, since Algorithm 1 has a computational complexity of $O(1)$, its execution time is negligible compared to executing the DL models. As such, we excluded its execution time from our calculations.

4 EXPERIMENTS

4.1 Experimental Setup

We evaluate SAGE on two edge, devices with significantly different computational capabilities: the Nvidia Jetson TX2 (TX2) and the industry-standard Nvidia DRIVE PX2 AutoChauffeur (PX2). The

TX2 is capable of 1.33 TeraFlops (TFLOPS) while operating within a power budget of 15 Watts (W). The more powerful PX2 is designed for real-world autonomous driving use-cases and has been used in vehicles such as the Tesla Model S [1]. It is capable of 8 TFLOPS within a power budget of 80 W. To serve as our cloud server, we used a Windows Desktop with an Nvidia 2080 Super, capable of 11.1 TFLOPS. It should be noted that, in a real-world deployment of SAGE, a more powerful cloud server could be used to increase the benefits of offloading.

In terms of the dataset, we use the CARLA conditional IL dataset from [8]. It contains RGB images in 200×88 resolution and control/sensor values extracted from the CARLA urban driving simulator [11]. We used the image data as well as the steering, accelerator, brake, and navigational command information from the dataset for training and evaluating the accuracy of both our original IL models as well as their bottlenecked counterparts. We implement and train our models in PyTorch to assess the difference in error between the original and *bottlenecked* models. To evaluate the model latency and energy consumption (T_i^{comp} and E_i^{comp} from Equations (7) and (8)), we directly obtained the measurements through the Caffe model timing API for the TX2. For the PX2 and cloud server, we used Nvidia's TensorRT library to compile and optimize the models for the hardware. TensorRT is designed to optimize the model architecture automatically (i.e., optimizing weights, parallelizing computations, combining redundant layers, etc.) to maximize inference performance on a given platform.

In our experiments, we evaluated both 1-camera and 3-camera implementations of our IL models. Our 1-camera experiments aim to demonstrate SAGE for a low-cost AV implementation consisting of either a TX2 or PX2 as the edge, device equipped with a single forward-facing camera. This implementation is practical for simple AV tasks such as adaptive cruise control, lane following, etc. Aligning with this goal, we evaluate the energy consumption and feasibility of SAGE with both low-resolution (88×200) and high-resolution (1280×720) camera data.

We also perform 3-camera experiments to demonstrate the feasibility of SAGE on more comprehensive AV hardware platforms. Multi-camera platforms are essential for real-world AV use cases such as urban/highway driving and point-to-point travel. Thus, we evaluated our IL models using three high-definition 1280×720 (720p) camera inputs on the PX2. Here, each model was modified to include three separate perception modules to process data from each camera. The outputs of the perception modules were then concatenated before being processed by the IL module, as was done in [16].

To evaluate the communication power cost needed in Equation (3), we use the transmitting and receiving power models derived in [19] for 3G, WiFi, and 4G LTE wireless technologies. Note that 5G energy evaluations are not available since we could not find any 5G-specific real-world power models in the literature as we found for the other technologies. In terms of the computation energy in Equation (8), we leverage the onboard sensing circuits within the TX2 board for estimating the execution and idle powers, whereas we use an external power meter for the PX2. We assume no packet losses in our evaluations, and as mentioned, we demonstrate SAGE's feasibility with widespread and currently available network technologies.

4.2 Performance Comparison of Original vs. Bottlenecked IL Models

Recall that the *bottleneck* acts as an encoder whose main purpose is to reduce the output data size to attain an efficient data transmission if needed. This data reduction is mainly achieved through reducing the number of *output channels* at the *bottleneck* layer (3 in the experiments). To give perspective, the *output channels* for any layer in any of the original DL models discussed here before introducing our alterations is 32, meaning that there is an $\approx 10\times$ reduction in output data size at least. To ensure that the introduction of a *bottleneck* into our models does not impact their

Table 2. Comparison between the Original IL Models and Their Modified Counterparts With *bottlenecks* After HND

Model	Mean Absolute Error (MAE)		
	Steering	Accelerator	Brake
IL-DenseNet-169	0.0177	0.0356	0.0129
IL-DenseNet-169 w/HND	0.0159 (−0.0018)	0.0509 (+0.0153)	0.0195 (+0.0066)
IL-ResNet-34	0.0259	0.0506	0.0199
IL-ResNet-34 w/HND	0.0263 (+0.0004)	0.0545 (+0.0039)	0.0259 (+0.0060)
IL-ResNet-50	0.0260	0.0514	0.0180
IL-ResNet-50 w/HND	0.0266 (+0.0006)	0.0601 (+0.0087)	0.0330 (+0.0150)
IL-CarlaNet	0.0259	0.0546	0.0228
IL-CarlaNet w/HND	0.0204 (−0.0055)	0.0589 (+0.0043)	0.0326 (+0.0098)

Values in parentheses are the differences in error between the models.

Table 3. Hardware Performance Metrics for Processing One 88 × 200 Camera Input

Network	Device	Power (W)			Latency (ms)			Energy (J)	
		E2E	Head	Idle	E2E	Head	Tail	E2E	Head
IL-DenseNet-169	Server	–	–	–	–	–	**2.238**	–	–
	TX2	5.446	5.430	1.659	215.543	**8.043**	207.5	1.1740	**0.0437**
	PX2	43.58	47.42	40.23	10.420	**1.112**	9.308	0.4541	**0.0527**
IL-ResNet-34	Server	–	–	–	–	–	**0.572**	–	–
	TX2	5.95	5.221	1.659	65.560	**11.612**	53.948	0.3901	**0.0606**
	PX2	46.99	47.51	40.23	4.534	**0.695**	3.839	0.2131	**0.0330**
IL-ResNet-50	Server	–	–	–	–	–	**0.607**	–	–
	TX2	5.682	5.415	1.659	89.231	**10.432**	78.799	0.5070	**0.0565**
	PX2	46.89	47.17	40.23	7.413	**1.195**	6.218	0.3476	**0.0564**
IL-CarlaNet	Server	–	–	–	–	–	**0.188**	–	–
	TX2	5.391	5.039	1.659	28.795	**8.727**	20.068	0.1552	**0.0440**
	PX2	45.54	46.33	40.23	1.659	**0.593**	1.066	0.0756	**0.0275**

E2E = processing the entire model end-to-end on the edge, device. Cloud server power/energy is ignored because this is not a constraint.

predictive performance, we evaluated the mean absolute error (MAE) of our models both with and without the bottleneck, shown in Table 2. In the cases with the bottleneck, we used HND to train the *head* of the bottlenecked model to mimic the original model's *head*, as described in Section 3.4. The results clearly show that the bottlenecked models perform very similar to the original models, with only a slight increase in MAE. For context, an MAE increase of 0.01 corresponds to a 1% increase in error between the model outputs and the outputs provided by the human driver.

4.3 Power, Energy, and Latency Evaluation on Hardware

From this point onwards, all IL models referred to are with *bottleneck* layers added. In Table 3, we compare the power consumption, energy consumption, and latency of different parts of the IL models on each hardware platform. By comparing the end-to-end (E2E) latency of the edge, devices with the edge, head latency and server tail latency, we see that offloading at the head provides ample time to account for network transmission latency. Furthermore, the table shows a significant energy reduction when processing the head model instead of the entire model end-to-end. These metrics illustrate the feasibility and potential benefits of our model.

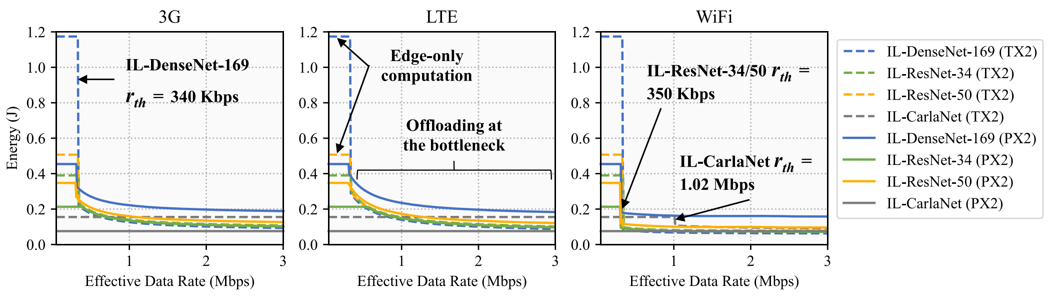

Fig. 6. Energy consumption of IL models developed through SAGE while processing a single 88 × 200 camera input at different data rates with 3G, 4G LTE, and WiFi. The transition point in each line occurs at r_{th}, which is when offloading begins at the *bottleneck*. Before this point, the energy consumption for the edge-only processing is ($E^{edge}_{head} + E^{edge}_{tail}$). After this point, the energy consumption is calculated as ($E^{edge}_{head} + E^{comm}_{i} + E^{edge}_{idle}$).

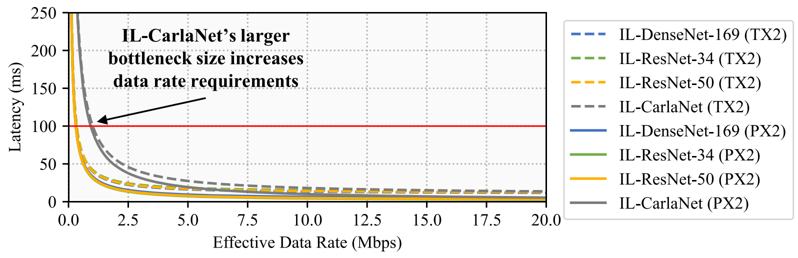

Fig. 7. End-to-edge latency of each model for offloading at the *bottleneck* for an AV with a single 88 × 200 camera input. The end-to-end latency includes edge, head processing latency, wireless network latency, and server processing latency at various network data rates. The red line indicates the 100 ms latency constraint.

4.4 Offloading Evaluation

4.4.1 Low Resolution. In Figures 6 and 7, we show results from evaluating models implemented through SAGE with a single 200x88 resolution camera input using the TX2 and the PX2.

Figure 6 shows the energy consumption of each IL model with each technology type at different values of effective data rate r^U. Recall that these values are obtained based on the offloading strategy in Section 3.5, where it is only feasible to offload when ($r^U_{i,j} > r_{th}$) and ($E^{comm}_i + E^{edge}_{idle} < E^{edge}_{tail}$). For each model, observe the sharp change in Figure 6 at r_{th} where the model switches from edge-only computation to cloud offloading. For IL-DenseNet-169, IL-ResNet-34, and IL-ResNet-50, this switching point occurs at approximately 350-400 Kbps on both the TX2 and PX2.

Although offloading can meet the latency constraint for some r^U values, the energy consumed by the networking components must still be considered. As shown in Figure 7, IL-CarlaNet can feasibly offload at 1 Mbps on both devices, but on 3G and 4G LTE, offloading consumes more power than edge-only computation. Thus, we only consider r^u values which are greater than r_{th}, at which offloading *saves* energy on the edge, device compared to edge-only processing. For IL-CarlaNet on the TX2, r_{th} is 7.7 Mbps on 3G, 3.62 Mbps on 4G LTE, and 1.02 Mbps on WiFi. This is likely because IL-CarlaNet has a larger data size at the *bottleneck* than the other models, increasing communication latency and energy. Interestingly, on the PX2, IL-CarlaNet's r_{th} is 13.66 Mbps for WiFi and there is no 3G or 4G LTE r_{th} under 100 Mbps for IL-CarlaNet that saves energy compared to edge-only computation. This is likely a result of the data size and the fact that IL-CarlaNet is a very

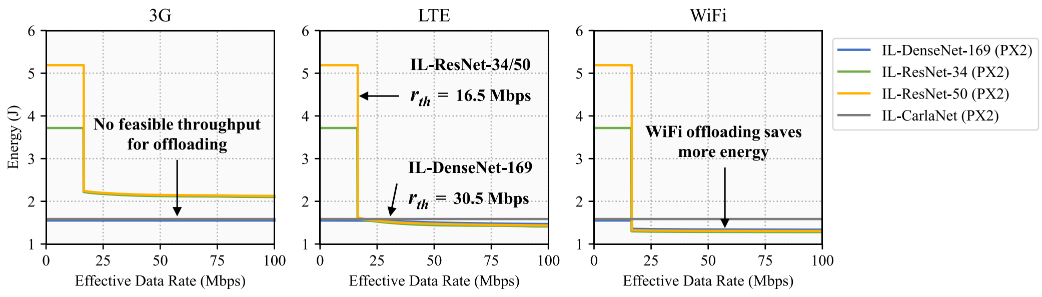

Fig. 8. Energy consumption of each model for processing a single 1280 × 720 (720p) camera input.

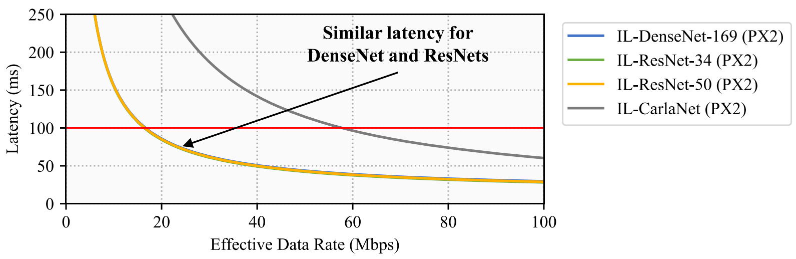

Fig. 9. End-to-end latency of each model for offloading at the *bottleneck* at different network data rates for an AV with a single 1280 × 720 (720p) camera input.

small model and the PX2 has a moderately high idle power consumption (40.23W), meaning that offloading would consume more power than simply running on the edge. Since IL-DenseNet-169, IL-ResNet-34, and IL-ResNet-50, are larger models, there is a clear benefit to offloading. Thus, the r_{th} remains at 320-390 Kbps. The only exception is IL-ResNet-34 using LTE on the PX2, which has an r_{th} of 550 Kbps, likely due to the efficiency of the PX2 compared to its idle power consumption and the network latency at this data rate.

Overall, when offloading at the r_{th} for each model and technology, the TX2 and PX2 consume an average of **49.78%** and **22.48%** less energy, respectively, compared to edge-only computation. Interestingly, when running edge-only, the PX2 consumes half as much energy as the TX2; however, when both offload at r_{th}, the PX2 consumes $\approx$ 25% more energy than the TX2. This is likely because the network latency outweighs the efficiency benefit of the PX2 at these low throughputs. Regardless, both devices significantly reduce edge, energy consumption by offloading.

For all models except IL-CarlaNet, the r_{th} is well within the operating range for all three network technology types. Figure 7 clearly shows that all models can meet the deadline of 100 ms with network throughputs as low as 320 Kbps. Above 15 Mbps, the benefit of higher data rates is minimal for this data size.

4.4.2 High Resolution. The previous experiment demonstrated that our approach is feasible and has significant benefits for low-resolution camera data. However, real-world AVs use high-definition cameras to improve perception performance and safety [1, 20, 21]. To emulate this application, we evaluate SAGE on camera data with a 1280x720 (720p) resolution, the resolution used for Tesla Autopilot 2.0 systems. We only assessed the PX2 on this application since it is infeasible for the TX2 to meet the deadline of 100 ms with this input size even when running on the edge, only. The results of this experiment are shown in Figures 8 and 9.

The larger input image size increases model sizes and data sizes at the *bottleneck* (59× larger for IL-CarlaNet and 47× larger for all other models), increasing the edge, processing latency, communication latency, and energy consumption significantly. This change is reflected in the figures, as IL-ResNet-34 and IL-ResNet-50 have an r_{th} of ≈ 16.5 Mbps. This data rate is well within the normal operating ranges of 4G LTE and WiFi connections. IL-DenseNet-169 has r_{th} values of 30.53 Mbps and 16.65 Mbps on 4G LTE and WiFi, respectively, but does not have a practical r_{th} under 100 Mbps for 3G. This is likely because 3G consumes significantly more energy to upload data than 4G LTE and WiFi. Also, TensorRT better optimized the IL-DenseNet-169 model since it consists of a large number of relatively small layers, reducing its energy consumption significantly compared to IL-ResNet-34 and IL-ResNet-50. This reduction decreases the potential benefits of offloading in this case. Compared to edge-only processing, offloading the models at r_{th} with 3G, 4G LTE, and WiFi reduces edge, energy consumption by **48.54%**, **41.96%**, and **50.72%**, respectively. With high-resolution data, the energy consumption benefit is more than double that of offloading low-resolution data, indicating that offloading is more beneficial for large, demanding edge, models.

Since the data size at the IL-CarlaNet model's *bottleneck* is 3.86× larger than that of other models, it requires a higher throughput (57.8 Mbps) than the other models to meet the deadline. Also, IL-CarlaNet's small model size reduces its edge, energy consumption, meaning that the communication energy consumption and idle power consumption could outweigh any potential savings. We found no r_{th} below 100 Mbps for any networking technology that reduces the energy consumption of IL-CarlaNet below that of edge-only processing.

4.5 Multi-camera Evaluation

State-of-the-art AVs use multiple high-definition cameras to capture more information about the vehicle's surroundings to improve decision-making, control, and safety [1, 16, 20, 21]. This problem is highly demanding in terms of energy consumption and network connectivity since the latency constraint remains the same at 100 ms despite the significant increase in input and model size. To evaluate SAGE on this application, we provide three 720p camera inputs to our models.

We adapt our models for this task by replicating the original 720p perception pipelines to form three parallel perception pipelines (one for each camera input). The outputs of these pipelines are then concatenated and passed to the IL portion of each model. Consequently, each of the parallel perception pipelines contains one *bottleneck* layer from which data can be offloaded. During offloading, we assume the data at all three *bottlenecks* are sent to the cloud simultaneously. To reduce the maximum throughput requirement in this application, we quantize the values at the *bottleneck* from 32-bit precision to either 16-bit or 8-bit precision before transmission. We tested IL-DenseNet-169 with quantizations of 16-bits and 8-bits at the *bottleneck* layer and found that the average difference in MAE compared to the original is just 1.6×10^{-10}, which is imperceptible. Thus, with 16-bit and 8-bit quantization, we reduce our throughput requirements by 50% and 75%, respectively, while having a negligible effect on performance. Once again, we only evaluate the PX2 in this application since the TX2 cannot meet the deadline of 100 ms with the 3-camera models.

In this application, all-cloud offloading approaches are entirely infeasible. Given that the input data size (three 720p images) is 8.29 MB total, they would require a minimum throughput of 664 Mbps to meet the 100 ms deadline. In contrast, the data size offloaded by our model with 16-bit *bottleneck* quantization is only 264 KB (31× smaller); with 8-bit quantization, this drops to 132 KB (62× smaller). In our experiments, we find that our approach is feasible at throughputs easily achievable by WiFi and 4G LTE. Our experimental results are shown in Figures 10 and 11.

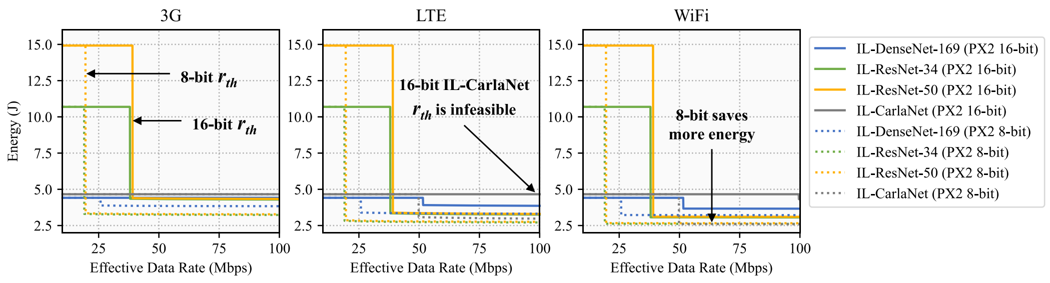

Fig. 10. Energy consumption of each model for processing three 1280 × 720 (720p) camera inputs. Results are shown for both 16-bit quantization and 8-bit quantization at the *bottleneck*.

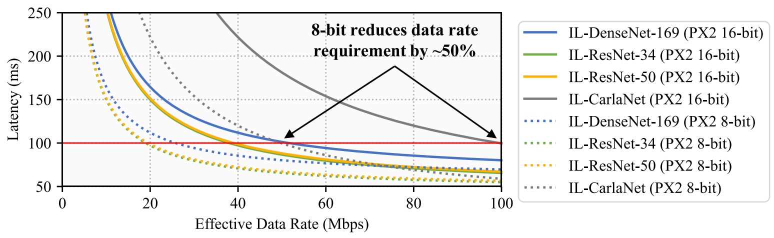

Fig. 11. End-to-end latency of each model for offloading at the *bottleneck* at different network data rates for an AV with three 1280 × 720 (720p) camera inputs. Results are shown for both 16-bit quantization and 8-bit quantization at the *bottleneck*.

As shown in Figures 11 and 10, with 16-bit quantization, IL-DenseNet-169, IL-ResNet-34, and IL-ResNet-50 can all offload at r_{th} values of 51.57 Mbps, 37.98 Mbps, and 39.05 Mbps, respectively. With 8-bit quantization, these r_{th} values drop to 25.79 Mbps, 18.99 Mbps, and 19.53 Mbps, respectively. With 8-bit quantization, most 4G LTE and WiFi connections can easily support the r_{th} data rates. Regarding 16-bit quantization, good quality 4G LTE and most WiFi connections should be able to support the r_{th} data rates [22]. On 4G LTE and WiFi, these models consume **52.67%** and **50.40%** less energy, respectively, by offloading at their r_{th} throughputs. The energy reduction is much more significant for IL-ResNet-34 and IL-ResNet-50 than IL-DenseNet-169, which we again attribute to TensorRT's model optimizations. It should be noted that, during offloading, all models appear to have very similar energy consumption. Practically, this means that an AV can run much larger models (e.g., use IL-ResNet-50 instead of IL-ResNet-34) without much difference in energy consumption provided a network connection with a data rate greater than r_{th} is available most of the time.

Once again, there is little benefit for offloading IL-CarlaNet due to the larger data size at the *bottleneck* (1.01 MB) and the relatively low energy consumption of the model running on the edge. With 16-bit quantization, IL-CarlaNet only saves energy on WiFi at a data rate above 99.51 Mbps. However, with 8-bit quantization, offloading becomes feasible for both 4G LTE and WiFi at 49.76 Mbps. Since IL-CarlaNet is a relatively small model, it may be better to run it on the edge, device most of the time and only offload on WiFi when network throughput is high.

5 DISCUSSION

In this section, we discuss our key findings from our experiments as well as the limitations, feasibility, and cost of SAGE. We also discuss future research directions.

5.1 Overall Findings

We found that SAGE was feasible for most IL models for all hardware configurations. By offloading at r_{th}, SAGE reduced edge, device energy consumption by **36.05%** with one low resolution camera, **47.07%** with one high-resolution camera, and **55.66%** with three high-resolution cameras. More energy could be saved by offloading at throughputs higher than r_{th} when possible. Additionally, our results indicate that SAGE saves more energy by offloading when input data sizes are larger (i.e., when using more cameras or higher resolutions). SAGE also reduces upload data size by **96.81%** and **98.40%** with 16-bit and 8-bit quantization, respectively, compared to directly offloading three 720p camera inputs. Besides, we found that our introduction of *bottleneck* layers only increased mean error by $\approx$ 1% and quantization had a negligible effect on error, meaning that SAGE could be scaled to even higher camera resolutions easily.

5.2 Limitations

In our experiments, we found that our offloading methodology was not particularly effective for IL-CarlaNet. With low-resolution data, it required a significantly higher r_{th} to provide a benefit than the other models; with high-resolution data and multiple cameras, there was no r_{th} below 100 Mbps that reduced energy consumption. In its current form, SAGE may not present useful offloading for small models and models with a proportionally large *bottleneck* size due to the increased energy cost of transmitting and receiving data compared to just running the entire model on the edge.

Additionally, although the 100 ms represents a reasonable worst-case bound, the current industry standard for real-time video processing is 30 frames/second, meaning that practically, the bound for completing the AV prediction task can be even tighter reaching $\approx$ 33 ms. From our experimental analysis, SAGE can meet this constraint when offloading the quantized version of the single full HD image data transformation. However, it fails to satisfy this requirement in the case of 3 HD camera inputs. Thus, experimentation with respect to AV industry-standard hardware and 5G wireless technology can provide a fair assessment of SAGE's capability to meet these tighter bounds.

Although our methodology has shown promise in terms of improving the overall performance efficiency, several other factors can impact the extent of this improvement given some real-world situations. It is possible that channel contention between users, packet loss, and channel coherence issues related to vehicle speed and environmental conditions could limit the benefits of our methodology. These effects are difficult to simulate accurately, so real-world experiments are still needed to gauge the energy savings offered by our methodology in these situations.

Lastly, we did not evaluate our approach on modular pipelines. However, since modular pipelines' perception modules generate the most latency [28], SAGE could be directly applied to these modules to achieve similar energy benefits. AV hardware platforms also handle other tasks such as route planning and user interfaces, but these applications constitute a minute part of the overall AV driving system. [28] has shown that the object detection, tracking, and localization modules (i.e., components of the modular version of the perception pipeline) comprise over 98% of the total computation, consuming 1.99 J per input. This proportion is very similar to the results we show in Table 1. Based on our energy savings with 3-camera offloading, if we introduce a *bottleneck* to the object detection module and offload the remaining modules to the cloud, we could reduce energy consumption from 1.99 J to 0.896 J, a savings of 55%.

5.3 Practicality and Cost

Since SAGE does not require any hardware modifications to the AV or network infrastructure, it is much more cost-efficient and flexible than other solutions such as ASIC design or 5G C-V2X/WAVE installation. The only added costs are those associated with hosting a cloud server to run the offloaded models. However, we demonstrated SAGE's feasibility with a Desktop PC as the cloud server, so hosting similar hardware in the cloud would likely be inexpensive. These costs could even be passed on to consumers, where a vehicle owner could elect to extend their AV driving range by paying for an offloading service as proposed in [41]. Compared to direct offloading, SAGE has significantly lower throughput requirements, making it much more practical for real-world deployment with the current networking infrastructure.

5.4 Future Work

In this work, we demonstrated the performance benefits attainable through the SAGE methodology over two NVIDIA hardware platforms, JETSON TX2 and DRIVE PX2. Although our approach is platform-agnostic, we intend to apply SAGE in our future works on different target hardware with different capabilities, like the high performance inference Neural Processing Units (NPUs) developed by ARM [4]. To ensure that our methodology does not introduce additional safety risks, it would also be prudent to evaluate each model on closed-loop evaluations in future work, such as judging each model's success rate at driving point-to-point in a simulator as in other works [7–9, 11]. Moreover, even though we demonstrated the merit of SAGE using the current prevalent network technologies, this research area is still relatively new, and problems such as energy optimization with multiple servers, modular AV architectures, and 5G networks remain unstudied. For example, SAGE can be adapted to address a multi-MEC server problem context. In this case, the action-space would expand from the AVs' perspective, for they would not only need to make an offloading decision each time step, but also identify which server should be selected for data transfer and task delegation. This would also entail additional dynamic factors to be considered, such as each server's load. Hence, a more sophisticated approach, like reinforcement learning [24], would need to be applied to solve the problem each time-step, in which previous connection experiences with the various servers could be leveraged through an in-place policy to guide the MEC server selection. These problems are left to be addressed in future works.

6 CONCLUSION

Designing AV control algorithms that are both safe and energy-efficient is a complex challenge that cannot be practically solved using simple direct offloading strategies. In this work, we proposed SAGE: a methodology for splitting the computation of IL end-to-end control models between the edge, and the cloud while minimizing network throughput requirements by adding *bottleneck* layers to the models. We evaluate SAGE on both large and small IL models and show that adding *bottleneck* layers only results in a minor performance impact. Our experiments demonstrate that SAGE reduces the edge, energy consumption of IL end-to-end control algorithms with both low-resolution and high-resolution camera data by **36.13%** and **47.07%**, respectively. Additionally, we show that SAGE is scalable to AVs that use three high-definition camera inputs, reducing energy consumption by **55.66%**, and can be practically implemented using current state-of-the-art AV hardware (PX2) and networking infrastructure (3G, 4G LTE, and WiFi). On all three applications, we demonstrate that the IL models can be offloaded at effective data rates that are well within the constraints of current network infrastructure while still meeting AV latency deadlines. We also find that the throughput requirements for offloading reduce by 50% and 75% when quantizing the *bottleneck* output to 16-bits and 8-bits, respectively, with a negligible change in model performance. Overall, we show that SAGE is practical for real-world, end-to-end control applications and can significantly curtail AV energy consumption.

REFERENCES

[1] 2016. All new Teslas are equipped with NVIDIA's new Drive PX 2 AI platform for self-driving-Electrek. https://electrek.co/2016/10/21/all-new-teslas-are-equipped-with-nvidias-new-drive-px-2-ai-platform-for-self-driving. (Oct 2016). [Online; accessed 9. Nov. 2020].

[2] Sam Abuelsamid. 2020. Nvidia cranks up and turns down its drive AGX orin computers. *Forbes* (Jun 2020). https://www.forbes.com/sites/samabuelsamid/2020/05/14/nvidia-cranks-up-and-turns-down-its-drive-agx-orin-computers.

[3] Mohammad Abdullah Al Faruque and Korosh Vatanparvar. 2015. Energy management-as-a-service over fog computing platform. *IEEE internet of things journal* 3, 2 (2015), 161–169.

[4] ARM. *arm npu Ethos-77*. Retrieved April, 2021 from https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-n77.

[5] Lei Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 2654–2662.

[6] Jang Hyun Cho and Bharath Hariharan. 2019. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[7] Felipe Codevilla, Antonio M. Lopez, Vladlen Koltun, and Alexey Dosovitskiy. 2018. On offline evaluation of vision-based driving models. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 236–251.

[8] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. 2018. End-to-end driving via conditional imitation learning. In *International Conference on Robotics and Automation (ICRA)*.

[9] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. 2019. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9329–9338.

[10] Mingyue Cui, Shipeng Zhong, Boyang Li, Xu Chen, and Kai Huang. 2020. Offloading autonomous driving services via edge computing. *IEEE Internet of Things Journal* 7, 10 (2020), 10535–10547.

[11] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. In *Conference on robot learning*. PMLR, 1–16.

[12] Stephan Eichler. 2007. Performance evaluation of the IEEE 802.11 p WAVE communication standard. In *2007 IEEE 66th Vehicular Technology Conference*. IEEE, 2199–2203.

[13] Jingyun Feng, Zhi Liu, Celimuge Wu, and Yusheng Ji. 2018. Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling. *IEEE vehicular technology magazine* 14, 1 (2018), 28–36.

[14] Alejandro González, Zhijie Fang, Yainuvis Socarras, Joan Serrat, David Vázquez, Jiaolong Xu, and Antonio M. López. 2016. Pedestrian detection at day/night time with visible and FIR cameras: A comparison. *Sensors* 16, 6 (2016).

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR 2016*. IEEE Computer Society, 770–778.

[16] Simon Hecker, Dengxin Dai, and Luc Van Gool. 2018. End-to-end learning of driving models with surround-view cameras and route planners. In *Proceedings of the european conference on computer vision (eccv)*. 435–453.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*. http://arxiv.org/abs/1503.02531.

[18] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely connected convolutional networks. *CoRR* abs/1608.06993 (2016).

[19] Junxian Huang et al. 2012. A close examination of performance and power characteristics of 4G LTE networks. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys'12)*. 225–238.

[20] Kyle Hyatt. 2019. Baidu unveils its camera-based Apollo Lite self-driving suite. *Roadshow* (Jun 2019). https://www.cnet.com/roadshow/news/baidu-apollo-lite-camera-based-self-driving.

[21] Kyle Hyatt. 2021. Argo gives its self-driving vehicle hardware a big upgrade. *Roadshow* (Jan 2021). https://www.cnet.com/roadshow/news/argo-self-driving-car-hardware-upgrade.

[22] Agbotiname Lucky Imoize, Kehinde Orolu, and Aderemi Aaron-Anthony Atayero. 2020. Analysis of key performance indicators of a 4G LTE network based on experimental data obtained from a densely populated smart city. *Data in brief* 29 (2020), 105304.

[23] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence between the cloud and mobile edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*. 615–629.

[24] M. Khayyat, I. A. Elgendy, A. Muthanna, A. S. Alshahrani, S. Alharbi, and A. Koucheryavy. 2020. Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks. *IEEE Access* 8 (2020), 137052–137062.

[25] Young-Duk Kim, Guk-Jin Son, Chan-Ho Song, and Hee-Kang Kim. 2018. On the deployment and noise filtering of vehicular radar application for detection enhancement in roads and tunnels. *Sensors* 18, 3 (2018).

[26] Peng-Yong Kong. 2020. Computation and sensor offloading for cloud-based infrastructure-assisted autonomous vehicles. *IEEE Systems Journal* 14, 3 (2020), 3360–3370.

[27] Xiaolu Li, Bingwei Yang, Xinhao Xie, Duan Li, and Lijun Xu. 2018. Influence of waveform characteristics on LiDAR Ranging Accuracy and Precision. *Sensors* 18, 4 (2018).

[28] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E. Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems.* 751–766.

[29] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. 2019. Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo'19).* Association for Computing Machinery, New York, NY, USA, 21–26.

[30] Y. Matsubara, D. Callegaro, S. Baidya, M. Levorato, and S. Singh. 2020. Head network distillation: splitting distilled deep neural networks for resource-constrained edge computing systems. *IEEE Access* 8 (2020), 212177–212193.

[31] Yoshitomo Matsubara and Marco Levorato. 2020. Neural Compression and Filtering for Edge-assisted Real-time Object Detection in Challenged Networks. (2020). arXiv:cs.CV/2007.15818

[32] Mohanad Odema, Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. 2021. LENS: Layer distribution enabled neural architecture search in edge-cloud hierarchies. In *2021 58th ACM/IEEE Design Automation Conference (DAC).*

[33] Nate Oh. 2017. NVIDIA Announces Drive PX Pegasus at GTC Europe 2017: Level 5 Self-Driving Hardware, Feat. Post-Volta GPUs. *AnandTech* (Oct 2017). https://www.anandtech.com/show/11913/nvidia-announces-drive-px-pegasus-at-gtc-europe-2017-feat-nextgen-gpus.

[34] Apostolos Papathanassiou and Alexey Khoryaev. 2017. Cellular V2X as the essential enabler of superior global connected transportation services. *IEEE 5G Tech Focus* 1, 2 (2017), 1–2.

[35] K. Samal, M. Wolf, and S. Mukhopadhyay. 2020. Attention-based activation pruning to reduce data movement in real-time AI: A Case-Study on Local Motion Planning in Autonomous Vehicles. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10, 3 (2020), 306–319.

[36] Kengo Sasaki, Naoya Suzuki, Satoshi Makido, and Akihiro Nakao. 2016. Vehicle control system coordinated between cloud and mobile edge computing. In *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE).* IEEE, 1122–1127.

[37] Ardi Tampuu, Tambet Matiisen, Maksym Semikin, Dmytro Fishman, and Naveed Muhammad. 2020. A survey of end-to-end driving: Architectures and training methods. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[38] Gregor Urban, Krzysztof J. Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. 2017. Do Deep Convolutional Nets Really Need to be Deep and Convolutional? (2017). arXiv:stat.ML/1603.05691

[39] Korosh Vatanparvar and Mohammad Abdullah Al Faruque. 2018. Design and analysis of battery-aware automotive climate control for electric vehicles. *ACM Transactions on Embedded Computing Systems (TECS)* 17, 4 (2018), 1–22.

[40] Xiufeng Xie and Kyu-Han Kim. 2019. Source compression with bounded DNN Perception Loss for IoT edge computer vision. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom'19).* Association for Computing Machinery, New York, NY, USA, Article 47, 16 pages.

[41] Ke Zhang, Yuming Mao, Supeng Leng, Sabita Maharjan, and Yan Zhang. 2017. Optimal delay constrained offloading for vehicular edge computing networks. In *2017 IEEE International Conference on Communications (ICC).* IEEE, 1–6.

[42] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. 2016. Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks. *IEEE access* 4 (2016), 5896–5907.

Received April 2021; revised June 2021; accepted July 2021