2023 IEEE International Conference on Edge Computing and Communications (EDGE) | 979-8-3503-0483-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/EDGE60047.2023.00026

Perception Workload Characterization and Prediction on the Edges with Memory Contention for Connected Autonomous Vehicles

Sihai Tang*, Shengze Wang*, Song Fu*, and Qing Yang*
*Department of Computer Science and Engineering
University of North Texas
Denton, Texas 76203, USA
Email: {SihaiTang, shengzewang}@my.unt.edu; {Song.Fu, Qing.Yang}@unt.edu

Abstract*—Vehicular Edge computing requires computational power from connected Edge devices in the network to process incoming vehicle work requests. This connection and offloading allows for faster and more efficient data processing and thus improves the safety, performance, and reliability of the connected vehicles. Existing works focus on the processor and its characterization, but they forgo the connecting components. Memory resource and storage resource is limited on Edge devices, and the two combined incur a heavy impact on deep learning. This is prominent as perception-based workloads have yet to be studied deeply. In our characterization, we have found that memory contention can be split into 3 behaviors. Each of these behaviors interacts with the other resources differently. Then, in our deep neural network (DNN) layer analysis, we find several layers that see computation time increases of over 2849% for convolutional layers and 1173.34% for activation layers. Through the characterization, we can model the workload behavior for the Edge based on the device configuration and the workload requirements. Through this, the impacts of memory contention and its impacts are quantified. To the best of our knowledge, this is the first such work that *characterizes the memory impacts towards vehicular edge computational workloads with a deep focus on memory and DNN layers.

***Index Terms*—Edge Computing; Deep Learning; Autonomous Vehicles; Object Detection; Workload Characterization; Memory Contention.**

I. Introduction

Perception is vital for Autonomous Vehicles (AV), and while AVs can be very safe, they are not perfect. While all car manufacturers aim for safety and efficiency when designing and making their vehicles, there will still be unforeseen issues. As the number one duty of an AV is to keep its passengers safe, the ability of the AV to make critical driving decisions based on the environment around it becomes paramount. In addition, the AV must also be able to process everything, from sensing to driving, in a timely manner to account for both its own driving actions as well as the actions of other entities around it.

Given this context, the main AV processing pipeline is the flow of data from sensors to perception, then to path planning before actuation. In such a pipeline, the module to detect objects is arguably the most important of them all, as plenty of news headlines and evidence shows [1]. Given the fact that object detection must be both accurate and fast, the most obvious choice for this module is some type of Deep Neural Network (DNN). With a solid history and a very active academic interest in the field, DNNs are widely researched and used in both academic and industry sectors. Deployments of DNNs for various tasks such as YOLO, SSD, Faster R-CNN for object detection [2], [3], [4], DeepLab for image segmentation [5] and LaneNet for lane detection [6].

These DNN workloads are commonly deployed as Edge workloads, however, for a DNN to be useful, it must be trained and optimized continuously to match the perils and new issues that inevitably rises from the advancement of time. A quick example would be the introduction of new climates or a variety of unmarked old roads. Due to this, many companies employ humans and semi-supervised methods to label and train the data that is generated from the vehicles out on the roads, with each vehicle generating close to 11 Terabytes worth of data daily [7].

In addition, while the Edge has much lower latency than the cloud, it also has much fewer resources as compared to the cloud. This constraint on available computing resources can make understanding perception workloads on the Edge nodes a critical step for vehicle and Edge computing [8].

As a vessel meant to carry people, the AV must also be safety critical with highly stringent requirements; latency and detection accuracy is at the forefront [9]. However, real-time requirements are notoriously difficult to satisfy, as seen by the sheer amount of research being done in this area as new technologies continuously push the boundaries [10]. Paired with the wide usage of DNNs throughout the perception scene, the crucial issue now becomes how can AVs and Edge nodes handle this workload without being constantly upgraded with the latest sensors and models.

As most AVs rely on their own data for real-time object detection, there are quite a few limits to what they can do. And the limits are particularly bad when it comes to sensor blind zones and obstructions [11].

In this paper, we analyze memory contention not only on CNN networks but also on individual layers; not only characterization but also prediction. To the best of our knowledge,

2767-9918/23/$31.00 ©2023 IEEE
DOI 10.1109/EDGE60047.2023.00026

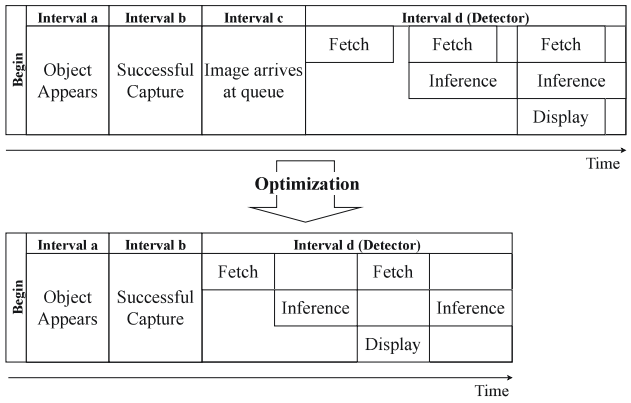

Fig. 1. Traditional approach to remove scheduling delays and memory contention.

this is the first work to deeply characterize the impact of memory contention on the performance of perception workloads on the edges. Existing studies focus on the CPU and GPU of the Edge as the limiting factor, but Memory is also a limiting factor often ignored.

Memory usage is a critical factor that can significantly impact the deployment of machine learning models for inference on the Edges. Machine learning models typically require significant amounts of memory to train and run, and the size of the model can grow quickly with increasing complexity. Therefore, deploying a machine learning model that uses a large amount of memory on a device with limited resources, such as an Edge device, can cause performance issues, such as slow execution or crashes. In an over-subsidized workload, where multiple tasks are simultaneously scheduled to overload the scheduler, only 4 tasks can be completed; despite each task having ample resources. Consequently, optimizing memory usage is crucial when deploying machine learning models for inference, as it can improve the performance and reliability of the system, reduce hardware requirements and cost, and increase the overall efficiency of the machine learning workflow. Therefore, finding ways that memory usage of a machine learning model impacts the workload at different levels is critical for various deployment and use cases.

In our analysis, we found that Memory, as a factor, can be generalized through characterization functions to model and predict workload behavior. This behavior holds true across different ML methods and on different platforms.

By deep analysis, we can extract the behavior and performance of every layer in a perception network. Then, we can monitor the behavior of individual layers of the model during the inference process under different constraints and conditions. This information is crucial for understanding the performance and behavior of the model and for optimizing its architecture and parameters. However, monitoring the behavior of individual layers can be challenging, especially when dealing with complex models with many layers.

Our research has shown that convolutional layers, along with Routing, Shortcut, and ReLU activation layers, are the predominant layers affected by factors such as memory availability. For example, when the memory is constrained, some layers may take longer to compute or may require more memory to operate efficiently, with some layers seeing a computation time increase of over 2849% increase for convolutional layers, over 1053% increase for Routing layers, over 1173.34% increase for ReLU, and over 271% for Shortcut layers. These findings are valuable for pinpointing memory bottlenecks in a perception network and developing dynamic memory allocation strategies to improve perception performance.

Our main contributions include:

- Characterizing and modeling Memory impacts on ML inference workloads for Edge devices
- Deep layer analysis pinpointing specific layers affected by over-subsidization and memory contention
- Generalization of the Edge components that affect ML inferencing workloads and define challenges for future research

The rest of the paper is structured as follows, in Section II we go over the background and motivation. In Section III, the profiling setup and variables are discussed, and in Section IV the results are shown and analyzed. Section V discusses the findings, and Section VI concludes the paper.

II. Background and Motivation

As the world adopts the new generation of sensor-laden vehicles, the default infrastructures need to follow suit. Modern vehicles, be they autonomous or semi-autonomous, requires complex and extensive computational resources to process the data it needs to keep their passengers safe. And to enable the safety of both the vehicle and its passengers, the computation needs to be real-time and spatially relevant. In works such as [7], the future landscape of connected vehicles is explored and the challenges clarified. In them, the challenge of offloading work away from the vehicle is one of the major concerns.

In recent years, the concept of what defines computing is blurred as many Edge devices such as Nvidia proprietary Edge, or even mobile phones can accomplish the same task. Edge computing is utilized in many fields such as unmanned aerial vehicles (UAVs) [12], or for enhancing the capabilities of Android-based devices [13]. Along that same line, Edge also is a big factor for connected autonomous vehicles (CAVs). In works such as [14], the inefficiencies for vehicle's onboard computation is clearly presented for CAVs, and the use of edge computing was used resolved the issues; the issues included network congestion, insufficient onboard vehicle computation resources and processing speed.

Exploring this topic in recent years, papers such as [11], [15] and [16] explore and prove the possibility of using Edge as a means of sensor fusion for CAVs. Then in works such as [17] and [14], the real-world applications of offloading to the Edge are studied. With the gates opened, numerous possibilities are highlighted, including the discussion of DNN

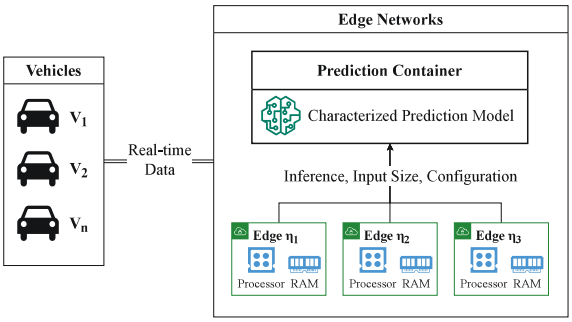

Fig. 2. Autonomous vehicle to Edge node interaction.

which has emerged as a crucial and highly visited topic on Edge, as indicated in [7].

The main workload for most AVs self-driving is based on DNN inferencing, and this can be any of the state-of-the-art models for object detection: YOLO [2], Faster R-CNN [4], Mask R-CNN [18], SSD [3], etc. While the aforementioned models do a great job, there are many trade-offs and optimizations to be had. For example, to increase the feasibility of deployment on lower-end computation hardware, improvements such as quantization, pruning, and architecture changes are often used [19], [20], [21].

Deploying machine learning models on Edge devices, especially low-end Edge devices, presents several challenges. One major challenge is the limited computing resource and memory resources available on these devices, which can restrict the complexity and size of the models that can be deployed. Additionally, Edge devices often have power constraints, which limit the amount of energy that can be used for computation.

This limitation becomes especially debilitating when it comes to the case of vehicular Edge computing. Most ML models require a large-weight model to function optimally, meaning that machine learning models deployed on Edge devices must be optimized for both power and speed, which can be difficult to achieve without sacrificing accuracy or performance.

Another challenge is the lack of standardization in hardware and software platforms, which can make it difficult to develop and deploy machine learning models that maintain consistency between AVs and vehicular Edge. Finally, the deployment of machine learning models on Edge devices can raise concerns about data privacy and security, as sensitive data from the vehicles may be stored and processed on the device itself. Addressing these challenges will require advances in hardware and software optimization, standardization, and data privacy and security measures [22], [23], [7].

Traditional machine learning and neural network improvements focus on overall latency per frame as well as the overall accuracy and optimizing the loss function. These generally focus on the CPU and GPU, but the impacts from Memory are not studied. In addition to the targeted hardware, methods such as the Pyramid Attention Network [24] or the Region Proposal Network [4] gave rise to many real-time detector algorithms. But these are generally developed without Edge resource constraints in mind.

Most commonly seen methods to get around the Edge memory constraints in literature usually sacrifice or explore tradeoffs. In works [25] and [26], the authors reduce the full YOLO architecture to reduce the computational load. Then in works such as [27], the workload is distributed through a pipeline rather than face the hardware constraints such as memory contention. Then in traditional methods such as [28], the memory contention for individual tasks is considered and removed, as seen in a reproduced Fig. 1.

Finally, modern literature that tries to tackle the Memory contention issues for DNN on Edge also sees limitations. In the host of memory-aware middleware works, we find MASA [29] along with many of the other memory-aware middleware works such as Deepeye [30], NestDNN [31], and DART [32]. These works approach the issue of CPU and Memory issues for low-end Edge devices from a middleware perspective, taking into account the layer impacts on the CPU and Memory respectively. However, even in works like MASA, the full impacts of memory are not fully examined. Also, due to the methodologies and hardware platform in MASA, the DNNs do not fully cover the single-stage and two-stage foundations; only two-layer types are analyzed. Additionally, the works mentioned above do not model the characteristics of memory as an impact variable, but rather facilitate the DNN process. Last but not least, most modern-day DNNs make use of Pytorch [33] as a backbone and framework, which was not included in these motivating works.

A. Challenges of Characterizing Machine Learning on Edge

Characterizing machine learning workloads on Edge devices presents several notable challenges. One major challenge is the diversity of Edge devices and their corresponding hardware configurations, which can make it very difficult to generalize performance and operation time metrics across different devices; energy consumption is also directly related to the workload intensity and processing time. Additionally, the complexity and variability of machine learning models, along with different input and network conditions, can make it difficult to predict how they will behave on different Edge devices, even if the other Edge devices are the exact same model and specification.

Furthermore, detailed profiling of machine learning workloads on Edge devices requires specialized tools and expertise that may not be readily available to developers or end users. In addition to the previously mentioned difficulties, the privacy of client data is another major concern as profiling workload requires monitoring the processing of sensitive data on the device itself. To address these concerns, more advances are in standardizing benchmarking tools or Edge privacy-preserving methods are needed as stated in surveys such as [34].

In works such as [35], [36], the memory and layer by layer connections to privacy are examined. Through layer by layer encoding, the papers show that it is possible to encrypt and process the data on a layer level without notable impacts to

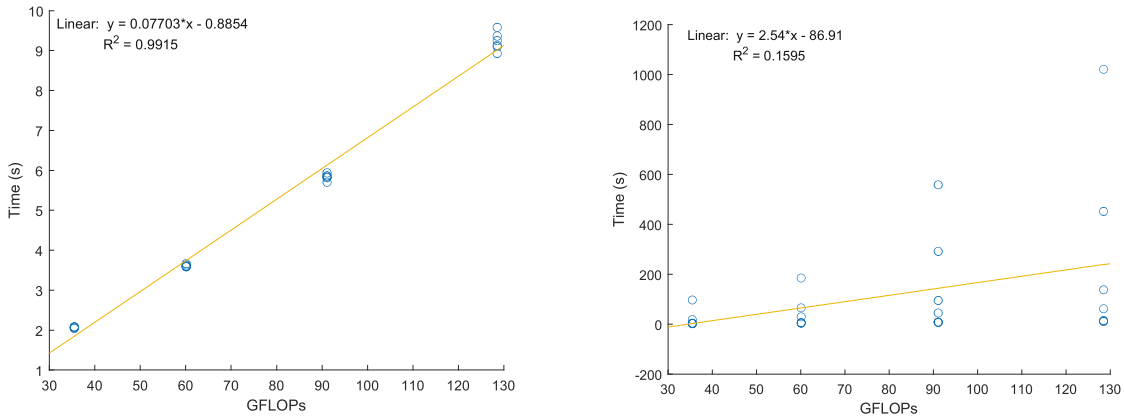

Fig. 3. Workload on single-stage perception With (Left)(a) and Without (Right)(b) Memory resource contention.

performance for CAVs. In the same line, traditional memory based privacy concerns also require deeper scrutiny.

III. Measuring and Characterizing Perception Workloads on the Edges

Deep learning-enabled perception networks can be broadly classified into two categories: single-stage and two-stage object detection. In this paper, we study both of them. In each category, we select the widely used and representative networks. Specifically, Single-Stage YOLO, both Darknet and Pytorch-based, and Two-Stage Faster-RCNN, Pytorch based. We represent YOLO as ω and Faster R-CNN as θ. By covering both single-stage and double-stage methods, we can attempt to generalize our findings to be applicable to other methodologies that utilize the same principles.

Our goal is to discover the uniqueness of each category in terms of resource demands, with a focus on the memory impacts in this paper, and performance bottlenecks arising from these resource demands.

In addition to the individual resource variable impacts, we also need to characterize the workload as a predictable model. For example, as Fig. 2 shows, we can utilize a generalized prediction model for the incoming workload to be able to facilitate a better scheduling module on the Edge. The edge network architecture would serve as both a communications and computational node for the CAVs accessing the edge network.

A. Profiling Setup

To profile our ML methods, we simulate the high-end Edge nodes with a machine equipped with an Intel Core i7-10750H, Nvidia GeForce RTX 2070, 16 GB of DDR4 RAM, and a 1 TB NVMe SSD. The total operating power constraint for the laptop is set to 250 Watts. For the lower-end Edge node, we opted for the Nvidia Jetson Xavier NX. It supports nine optimized power budgets to cap the CPU core numbers and their frequencies. Power modes in our experiments include 20W, 15W, or 10W TDP with six, four, or two CPU cores.

B. Features and Formulation

To define how incoming AV workloads will impact available Edge nodes, there are two sets of variables to consider, the set of inputs from AV and the set of resources from available Edge nodes. We define the set of AV as $V = \{ v_1, v_2, \dots \}$ and the set of Edge nodes as $E = \{ \eta_1, \eta_2, \dots \}$. Where $v_i = \{ v^i_{data}, v^i_{task} \}$ and $\eta_i = \{ \eta^i_{conf}, \eta^i_{queue} \}$ are monitored and facilitated by the Prediction Module. Following the input requests from the vehicles in Fig. 2, we allow for the Edge nodes in the service range to process the incoming tasks. Then, the individual Edge nodes are able to determine the request and workload from the incoming requests. The individual nodes can then communicate the task requirements and their own resources to the Edge manager, which can then quickly direct the workload to an Edge node that can finish the task within an acceptable deadline.

Next, to formulate potential Edge configurations, we define the potential variables as follows:

- Processor resource
- RAM as the memory resource
- Workload size calculated from the requested service and the input size
- Time as the quantifying metric

IV. Analysis and Key Insights

The difference in Edge deployment hardware and task type brings up a challenging issue. Different AV workload requests and variations in hardware performance on the Edge nodes can be hard to plan for. In turn, this negatively impacts the reliability of DNN inferencing and brings the essential challenge to robust workload characterization.

A. Memory and Computation Load

In Section II, we discuss how memory contention can be a significant influencing variable when time or performance latency is crucial. Based on Fig. 3, we see two distinctive states, highlighting the importance of managing memory contention in such scenarios.

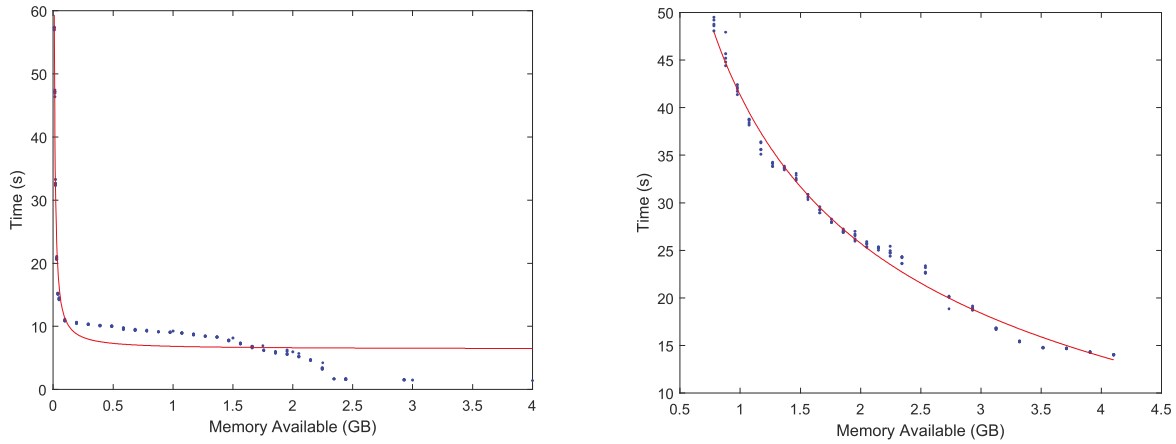

Fig. 4. Memory Contention on a low-end Edge between single-stage and two-stage perception. Left(a) represents YOLO and Right(b) represents Faster R-CNN

In our profiling, we quantify three states for the memory variable.

- Scenario 1: No Memory resource contention
- Scenario 2: Memory resource contention typically encountered under heavy workload
- Scenario 3: Extreme Lack of Memory resources

In Scenario 1, memory resource is plentiful for all of the tasks with no bottlenecks for any reason. In Scenario 2 however, memory resource is strained by either access racing conditions or resource availability. Temperature and system-bus bottlenecks are not considered in Scenario 2 as the system itself will be unstable for workloads. Finally, Scenario 3 covers the other possible memory allocation constraints, spanning from low to the lowest possible operational limits.

While in Scenario 1, Fig. 3.a, it is clear that the model does not deviate from the CPU variable-dominated linear model, and we can formulate the model as:

$$\eta_{time} = 0.077 * \gamma - 0.89 \tag{1}$$

where $\gamma = GFLOPs$ represent the amount of computation required by the workload.

However, if applying Equation 1 to the same set of workloads with the added memory limitations, the model will not hold true as seen in Fig. 3.b. While traditional methods of optimization and scheduling will alleviate this issue, they will not be able to solve it completely.

And so, to accurately characterize the impact of memory contention across various platforms, more characterization is required for workloads falling under Scenario 2 and 3, which experience moderate to heavy memory contention.

By modeling the behavior of the algorithm under different resource constraints, we can profile and characterize the impacts of Memory contention, where Faster R-CNN is referred to as (θ) and YOLO as (ω). Starting off, the max amount of RAM required to run each ML method is profiled, and we represent this value as θ_{MAX} for Faster R-CNN and ω_{MAX} for YOLO. Each ML is profiled with sequentially less available RAM available until task failure.

Starting off, in Fig. 5, we profile the workload characteristic under Scenario 3 and clearly see a direct relation with the amount of memory available. Based on the data gathered, we can model the two ML workload behaviors as Equations 2 and 3 respectively. CPU Frequency is represented as $\nu = CPUPower$ and Memory is represented as $\beta = Memory$. It should be noted here that while the R^2 is acceptable at a minimum of 0.9, some extreme cases such as operating ω at 100MB or less will have a higher variance when predicted with this model.

$$\eta^{\theta}_{time} = 0 * \nu + 1016 * \beta^{-0.06054} - 930 \quad \text{with } R^2 = 0.9041 \tag{2}$$

$$\eta^{\omega}_{time} = 0 * \nu + 0.2379 * \beta^{-1.03} + 11.38 \quad \text{with } R^2 = 0.9560 \tag{3}$$

To fully encompass Edge workload parameters for Scenario 3, and to ensure the character is consistent across different platforms, we analyze the same scenario on the Jetson device, shown in Fig. 4. Here, while the device is able to take smaller increments and holds less variance in the data, the behavior is much the same. Based on the data, the second set of models derived from Faster R-CNN(θ) and YOLO(ω) are represented in 4 and 5.

$$\eta^{\theta}_{time} = 0 * \nu + 42.418 * \beta^{-0.768} \quad \text{with } R^2 = 0.9837 \tag{4}$$

$$\eta^{\omega}_{time} = 0 * \nu + 6.386 * \beta^{-0.413} \quad \text{with } R^2 = 0.8694 \tag{5}$$

With the Jetson Memory model, we are able to predict the execution time with an $R^2 = 0.98$ for θ and a $R^2 = 0.87$ for ω.

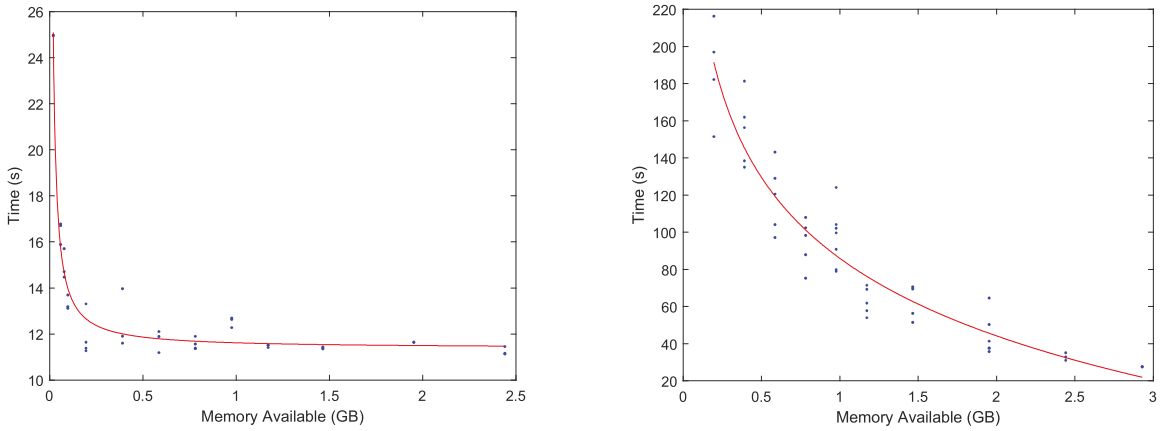

Fig. 5. Memory Contention on a high-end Edge between single-stage and two-stage perception. Left(a) represents YOLO and Right(b) represents Faster R-CNN

This is great for both industrial cost estimation of how many incoming tasks an Edge node can handle. With the memory impacts characterized, we now look at how it impacts each individual layer.

B. Layer Wise Characterization

With the insights from memory profiling, we can examine the layer information for further analysis. First, both of our ML methods use some common layers, such as convolution layers, activation layers, and operations such as max-pooling. In Fig. 7, we show the difference between the layers in a normal operation versus a resource-constrained operation. It is clearly evident the impacts of memory contention affects only selected layers of the architecture.

Given the shared building block layers, we profile the layer-by-layer workload for both aforementioned scenarios. The layers that are impacted are shown in Fig. 8 for YOLO, and Fig. 9 for Faster R-CNN. Upon deep analysis of the data gathered, we can see that the most heavily impacted layers are at the beginning of the architecture for both ω and θ. Table I highlights the layers most impacted for YOLO, and table II highlights the layers for Faster R-CNN.

YOLO Layer	Normal	Contention	Percentage Increase
0 Convolution	0.2705s	0.4792s	77%
1 Convolution	0.5176s	14.311s	2665%
6 Convolution	0.5046s	14.879s	2849%
9 Routing	0.0108s	0.1249s	1053%
10 Convolution	0.2669s	0.5167s	93%
11 Convolution	0.4157s	0.7581s	82%
16 Convolution	0.2113s	0.2816s	33%
20 Shortcut	0.0015s	0.0057s	271%

TABLE I
SINGLE-STAGE PERCEPTION CNN LAYERS WITH AND WITHOUT MEMORY CONTENTION.

Starting with Fig. 8.a, the data shows two layers, 1 and 6, that show heavy impact from memory contention. Both layers are convolutional layers with a calculated 3.407 GFLOP

Faster R-CNN Layer	Normal	Contention	Percentage Increase
3 Relu Activation	1.8892s	22.1668s	1173.34%
5 Convolution	0.5995s	3.4108s	468.9%
7 Convolution	1.1588s	28.272s	2339.77%
12 Convolution	0.9201s	6.551s	12.96%
14 Convolution	0.9158s	9.6611s	954.93%

TABLE II
TWO-STAGE PERCEPTION CNN LAYERS WITH AND WITHOUT MEMORY CONTENTION.

theoretical computational load, but whilst this load seems big, similarly sized loads are spread throughout the architectures in other layers as well. Layer 1 showed a 27.65 times increase and Layer 6 showed a 29.49 times increase over the non-contention scenario. This gives big insight as to where the most affected areas are for the YOLO architecture.

However, in Fig. 8.b, with the two layers from Fig. 8. a removed, we see other smaller impacted layers. When diving deeper, we find that in our profiling results, layers 9 and 20 show an abnormal amount of increase compared to both its theoretical GFLOP value as well as the non-contention latency. For Layer 9, a routing layer with GFLOP of near 0, the memory contention caused the latency to increase by over 1053%. Similarly, for Layer 20, a shortcut layer also with a near 0 GFLOP of 0.001, we see a latency increase of 271%.

It is logical for the initial convolutional layers to be heavily impacted by the lack of memory due to the input needing to be loaded from storage, but our insight on the impacts on layers 9 and 20 opens up potential paths for future research.

Next, we analyze the layer data for Faster R-CNN in Fig. 9. Here, the data shows 5 layers that are affected the most, and of those, layers 3 and 7 stand out. Layer 3 is a relu activation layer with a 1173% increase and layer 7 is a convolutional layer with a 2339.77% increase.

C. Compute Power and Processor Characterization

On both our platforms, we started with a fixed input load for both ML methods. In Fig. 6, we show the performance

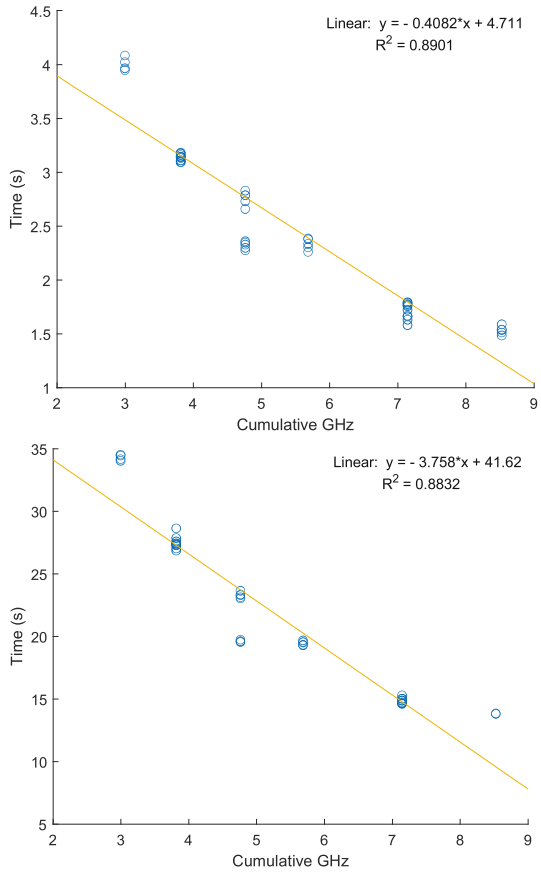

Fig. 6. Characterizing workload on Jetson with different CPU configurations. With single-stage perception network YOLO (Top)(a), and two-stage perception network Faster R-CNN (Bottom)(b).

of Faster R-CNN on 4 different CPU power Thermal Design Power (TDP) and Core configurations, with inference time as the quantifying metric for characterization of workload. From the data points gathered in Fig. 6.b, we model the Scenario 1 of Faster R-CNN impact as follows, where $\nu = CPU$:

$$\eta^{\theta}_{time} = -3.758 * \nu + 41.62 \quad \text{with } R^2 = 0.8832 \tag{6}$$

In the same fashion, we have Fig. 6.a showing the same profiling results for YOLO on the same CPU power and core configurations. Again, the model for the Scenario 1 impacts on YOLO Workload can be formulated as follows:

$$\eta^{\omega}_{time} = -0.4082 * \nu + 4.711 \quad \text{with } R^2 = 0.8901 \tag{7}$$

When combined with the memory variable from Scenario 2, we get the Scenario 2 Equation 8 for Faster R-CNN With memory measured in Gigabytes and Frequency measured in Gigahertz:

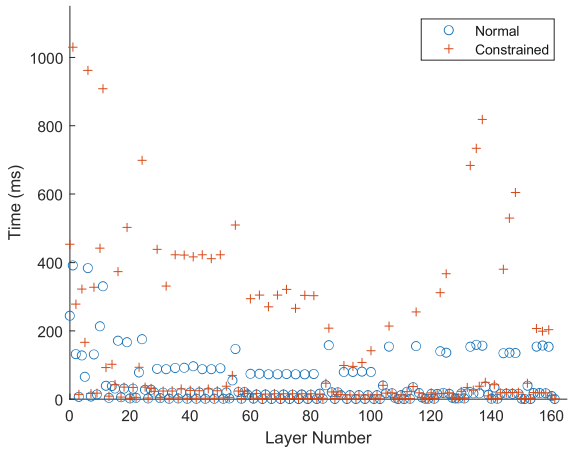

Fig. 7. Layer information for single-stage perception full architecture for both no contention and under oversubsidized contention.

$$\eta^{\omega}_{time} = 97.201 - 9.335 * \beta - 5.862 * \nu \quad \text{with } R^2 = 0.927 \tag{8}$$

In the same fashion, when Characterizing the entirety of the workload, we have Scenario 2 represented as Equation 9 for YOLO, we have the following equation:

$$\eta^{\delta}_{time} = 23.713 - 3.506 * \beta - 1.361 * \nu \quad \text{with } R^2 = 0.924 \tag{9}$$

D. Workload Size and Resource requirement

In the CPU profiling, we find that there is a direct linear relationship between the CPU computation power and the inference performance for both models, as shown in Fig. 6.

But the input size of the task can also determine how much computational power is needed. For example, processing higher-resolution sensor data requires much more computational power as opposed to a low-resolution sensor. To characterize this, we conducted profiling with different input sizes as shown in Fig. 10.

However, when we introduce another variable, memory, to the equation, then the profiling results change drastically as we can see in Fig. 3. When we apply the direct linear formula with adjusted constants to Fig. 3.b, the R^2 fit for the formulation based on the CPU variable becomes progressively inaccurate.

E. Insight Analysis and Generalization

With the data from our profiling on which variables affect ML workloads, we can now try to generalize our prediction model.

Based on the key insights from formulas 2, 3, 4 and 5, we can consolidate the findings for Scenario 3 in the following manner:

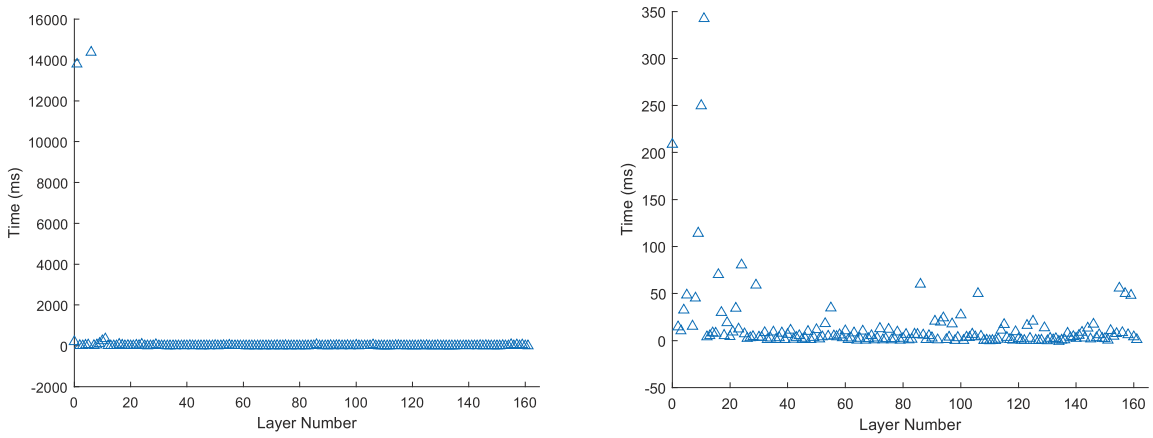

Fig. 8. Single-stage perception full architecture layer impacts of memory contention breakdown.

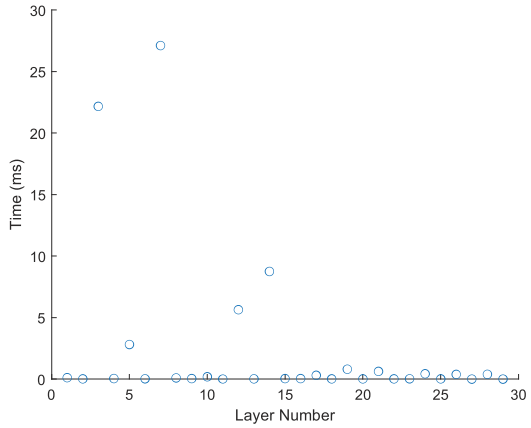

Fig. 9. Two-stage perception CNN layers with memory contention.

$$\begin{aligned} \eta_i^{\theta} &= k_i * \beta^{a_i} + Q_i \\ \eta_j^{\omega} &= k_j * \beta^{a_j} + Q_j \\ &\dots \end{aligned} \tag{10}$$

where k_i, a_i indicates the variable coefficient constant bound to each Edge node, Q_i as the respective constant value, and β reflects the currently available memory. The removal of the variable ν is needed in Equation 10 to accurately reflect Scenario 3. We can address the variance of thermal limitations through the rolling window average in 11, where n indicates a mark of time, thus allowing for accuracy. Here, the rolling window for k_n and a_n can be set so that $k_1 = k_2 \dots k_n = k_{n+1}$ where $n+1$ indicates the current time constant for Equation 10.

$$\begin{aligned} k_n &= \frac{k_1 + k_2 + k_3 + \dots + k_n}{n} \\ a_n &= \frac{a_1 + a_2 + a_3 + \dots + a_n}{n} \end{aligned} \tag{11}$$

By combining the two characterizing models, we can then formulate the decision tree for the running task queue c on the Edge, where $\beta_i^{threshold}$ for task i represents the memory threshold for Scenario 3, where the CPU variable impact is dwarfed by the memory impacts. This threshold varies based on the ML algorithm, and in our results, rested around 100MB for YOLO and 800MB for Faster R-CNN. The decision tree for predicting workload is as follows:

$$c\text{: } \{1, \dots, n\} \text{ where } \beta_{Available} = \beta_{Max} - \sum_{i}^{n} c_i$$

such that processing time for task: a_i in the task queue c can be calculated as follows

$$\begin{cases} c(a_i) = \text{Scenario 1} & \text{for } \beta_i^{threshold} > \beta_i^{Required} \\ & \quad \&\& \ \beta_i^{Required} < \beta_i^{Available} \\ c(a_i) = \text{Scenario 2} & \text{for } \beta_i^{threshold} > \beta_i^{Required} \\ & \quad \&\& \ \beta_i^{Required} > \beta_i^{Available} \\ c(a_i) = \text{Scenario 3} & \text{for } \beta_i^{threshold} < \beta_i^{Required} \end{cases}$$

From the profiling results, the first key insight is the impact of low memory resources for a given workload. Traditional CPU-based scheduling approach does not address the physical memory limitations, and thus cannot account for situations such as over-subsidization of an Edge node. In addition, the amount of data required for a simple predictive model based on our insight is very small and thus can be customized quickly and efficiently.

We have identified the characteristics of the various Edge node components relevant for CAV-based inference workload, and classify the variables for CPU and Input size as linear while taking special measures for over-subsidization situations.

Deeper profiling into the layer performance of the data shows that it is only a few specific layers that are affected by the memory contention. To start off, in Fig. 8 and Fig. 9,

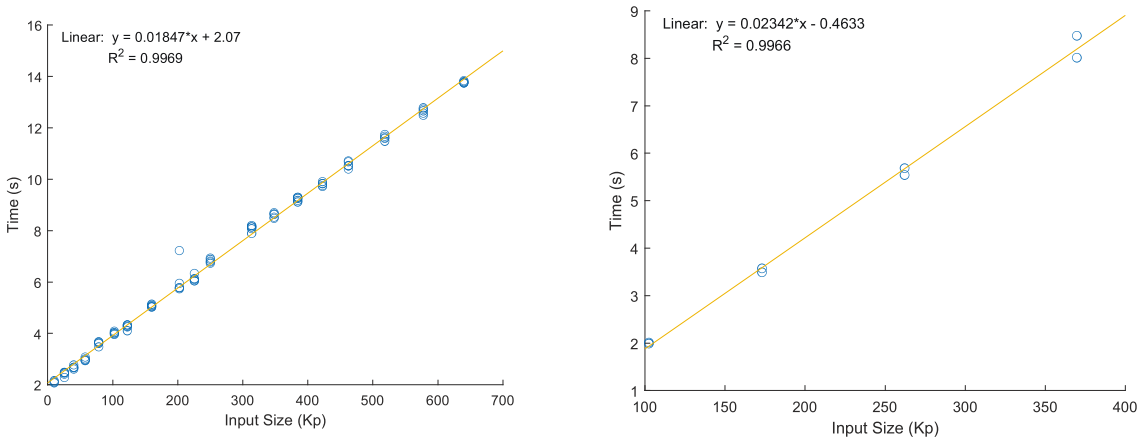

Fig. 10. Edge input vs time for YOLO and Faster R-CNN.

we see the impacted layers for ω and θ respectively. These findings allow for a much more targeted approach to research and optimize ML methods for Edge devices with expectations of memory contention or lower hardware specifications.

V. Implications and Discussion

In our generalization, we first considered a non-linear approach to encompass all the variables from Section IV at the same time. To do this, we test out multiple modeling methods. We show the results of including all of our targeted variables in Table I. Here, we see that while Linear Regression does give a decent model accuracy, it is out shadowed by methods such as MLP [37] and Correlated Nystrom Views [38].

Method	Model Accuracy
Linear Regression	0.82
Gaussian Process	0.76
Isotonic Regression	0.85
MLP* Regressor	0.88
MLP Base	0.84
Pace Regression	0.8
Radial Basis Network	0.43
Radial Basis Regressor	0.87
SMOreg (SVM)	0.81
Correlated Nystrom Views	0.9

TABLE III
Traditional and ML predictor models based on the features listed in table I. Coefficients of layers point to equation (1).

Through further testing, we have tried to train and tune the aforementioned models based on actual data and compare their prediction capabilities across platforms. But due to the data sample size, the models see overfitting issues. In an attempt to get around this, we used SMOTE [39] to upsample our data where needed to achieve a better-generalized result. However, the effort was too tedious and require further research to be optimized.

While we believe that the ML method of modeling based on the variables will outperform the generalized formulations presented, it will require much more data to be sufficient and will require retraining for each individual Edge node in a self-update loop, which will also require more resources.

A. Unusual findings

There are several unusual findings that are interesting. First, in our profiling, we find that oversubsidized workloads will trigger the OOM killer Linux kernel on the Nvidia Jetson, but not on the other platforms with the same Linux kernel and OOM policy. As this exceeds the scope of this paper, we did not investigate this phenomenon further. This finding may be pursued further as a CAV workload scheduling optimization issue.

Another anomaly we encountered during profiling was the discrepancy between the theoretical computation load and the actual workload latency profiled. For example, as explained in the layer-wise profiling section, we encountered a situation where shortcut layers and routing layers are having larger latency than normal convolutional layers. While this can be directly linked to the lack of memory, but it does not explain the huge latency increase we profiled. Also, in computation size, both shortcut layers and routing layers have a very small computation footprint, which makes this finding even more useful for optimization research. Again, as the technical challenges found in this phenomenon exceed the scope of this paper, this was not investigated further. We hypothesize that this may be a direct cause of a linear or sequential architecture workflow. Future works for this would be to investigate the base ML algorithms and also ones that can be paralleled, such as the inclusion of transformers, to see if a method can be used to allow for a consistent theoretical to real world performance translation.

While these findings did not affect the workload profiling, they do present additional avenues for future research.

VI. Conclusion and Future Work

Machine learning and AI workloads have achieved stability in our current society, with the widespread adoption of Edge technology by major technology companies like Microsoft

Azure, Amazon AWS, and IBM [40], [41], [42]. Despite the benefits, utilizing on-road Edge to offload workloads from CAVs presents numerous unknowns. Existing works on optimizing and facilitating machine learning on Edge devices lack the deep insights required for Edge users and device providers to estimate the behavior of machine learning workloads across diverse configurations. In this paper, we address these issues and identify potential areas for improvement. We propose a novel approach to optimize workloads by considering Edge device parameters, workload input, and algorithm architecture. The profiling and insights provided in our approach can be generalized beyond the specific machine learning algorithms examined in this paper.

While the models we present are basic, they offer better clarity in characterizing workload behavior compared to traditional machine learning (ML) models. Although some ML models like MLP and Correlated Nystrom Views can predict workload behavior, they necessitate extensive training and struggle to encompass different ML inference algorithms.

By characterizing and generalizing our findings, we provide valuable insights into the performance and potential of Edge devices for machine learning workloads. Our analysis demonstrates that even when overloaded with memory contention, algorithms such as YOLO and Faster R-CNN can still complete their tasks, suggesting that resource oversubscription on Edge devices is a viable option for non-real-time sensitive tasks.

Moreover, leveraging memory and layer-wise information enables optimization of both the architecture and deployment of machine learning models, leading to improved efficiency of Edge devices and reduced energy consumption. Our analysis identifies critical bottlenecks in machine learning workloads, as discussed in Sections IV and V.

The ability to predict the execution time of specific layers offers several benefits for future research. Firstly, it allows for the optimization of model architecture and hyperparameters by identifying the most time-consuming layers and optimizing their performance. Secondly, it enhances the efficiency of the model during training or inference by scheduling the layers to maximize available resources and reduce overall execution time. Finally, it facilitates the development of new algorithms and techniques that consider the performance characteristics of individual layers, enabling the creation of more efficient and accurate machine learning models.

The contributions of our paper are significant. Our proposed approach provides a targeted method for predicting the performance and maximum workload capabilities of Edge devices for machine learning inferencing tasks. These insights enable other researchers and service providers to conduct future research on more efficient algorithms and strategies for resource management, including memory and processing power, ultimately reducing contention, and improving overall performance. Additionally, exploring the utilization of distributed systems and Edge-to-cloud architectures can help alleviate resource contention and enhance scalability.

Another promising area to explore based on our findings is the development of accurate and efficient profiling tools and methodologies capable of handling the complexity and variability of machine learning models and Edge device hardware. Through a thorough understanding of workload behavior, standardized benchmarks and metrics can be established to target variables for more generalized use cases. Furthermore, our findings suggest the need to address privacy considerations. While our profiling did not account for privacy algorithms and methods applied to the selected workloads, incorporating privacy-preserving techniques and data obfuscation methods may reveal different workload characteristics and yield additional insights. Considering privacy aspects in workload profiling can provide a more comprehensive understanding of the challenges and opportunities involved.

In conclusion, there are numerous potential areas for future research in profiling machine learning workloads on Edge devices. Addressing these challenges will be crucial for enabling the widespread deployment of machine learning models on Edge devices. By delving into these areas, researchers can contribute to the advancement of efficient algorithms, resource management strategies, profiling tools, and privacy-preserving techniques, ultimately facilitating the seamless integration of machine learning in Edge computing environments.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their constructive comments and suggestions. This work has been supported in part by the U.S. National Science Foundation grants CNS-2231519, CNS-2113805, CNS-1852134, OAC-2017564, ECCS-2010332, CNS-2037982, DUE-2225229, and CNS-1828105.

REFERENCES

[1] "Exclusive: Surveillance footage of tesla crash on bay bridge," https://theintercept.com/2023/01/10/tesla-crash-footage-autopilot/.

[2] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[5] 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 transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[6] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool, "Towards end-to-end lane detection: an instance segmentation approach," in *2018 IEEE intelligent vehicles symposium (IV)*. IEEE, 2018, pp. 286–291.

[7] S. Lu and W. Shi, "Vehicle computing: Vision and challenges," *Journal of Information and Intelligence*, 2022.

[8] A. Waheed, M. A. Shah, S. M. Mohsin, A. Khan, C. Maple, S. Aslam, and S. Shamshirband, "A comprehensive review of computing paradigms, enabling computation offloading and task execution in vehicular networks," *IEEE Access*, vol. 10, pp. 3580–3600, 2022.

[9] L. Liu, Z. Dong, Y. Wang, and W. Shi, "Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 305–317.

[10] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, and A. Taherkordi, "Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[11] Q. Chen, X. Ma, S. Tang, J. Guo, Q. Yang, and S. Fu, "F-cooper: feature based cooperative perception for autonomous vehicle edge computing system using 3d point clouds," in *ACM/IEEE Symposium on Edge Computing (SEC)*, 2019.

[12] H. Sun, B. Zhang, X. Zhang, Y. Yu, K. Sha, and W. Shi, "Flexedge: Dynamic task scheduling for a uav-based on-demand mobile edge server," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 15 983–16 005, 2022.

[13] Y. Yao, B. Liu, Y. Zhao, and W. Shi, "Towards edge-enabled distributed computing framework for heterogeneous android-based devices," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, 2022, pp. 531–536.

[14] S. Tang, B. Chen, H. Iwen, J. Hirsch, S. Fu, Q. Yang, P. Palacharla, N. Wang, X. Wang, and W. Shi, "Vecframe: A vehicular edge computing framework for connected autonomous vehicles," in *2021 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2021, pp. 68–77.

[15] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Collaborative data scheduling for vehicular edge computing via deep reinforcement learning," *IEEE Internet of Things Journal*, 2020.

[16] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[17] S. Hu, G. Li, and W. Shi, "Lars: A latency-aware and real-time scheduling framework for edge-enabled internet of vehicles," *IEEE Transactions on Services Computing*, vol. 16, no. 1, pp. 398–411, 2023.

[18] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[19] Y. Cai, W. Hua, H. Chen, G. E. Suh, C. De Sa, and Z. Zhang, "Structured pruning is all you need for pruning cnns at initialization," *arXiv preprint arXiv:2203.02549*, 2022.

[20] Z. Hou, M. Qin, F. Sun, X. Ma, K. Yuan, Y. Xu, Y.-K. Chen, R. Jin, Y. Xie, and S.-Y. Kung, "Chex: channel exploration for cnn model compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 287–12 298.

[21] S. Cao, L. Ma, W. Xiao, C. Zhang, Y. Liu, L. Zhang, L. Nie, and Z. Yang, "Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 216–11 225.

[22] S. Parkinson, P. Ward, K. Wilson, and J. Miller, "Cyber threats facing autonomous and connected vehicles: Future challenges," *IEEE transactions on intelligent transportation systems*, vol. 18, no. 11, pp. 2898–2915, 2017.

[23] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2019.

[24] H. Li, P. Xiong, J. An, and L. Wang, "Pyramid attention network for semantic segmentation," *arXiv preprint arXiv:1805.10180*, 2018.

[25] D. P. Carrasco, H. A. Rashwan, M. Á. García, and D. Puig, "T-yolo: Tiny vehicle detection based on yolo and multi-scale convolutional neural networks," *IEEE Access*, 2021.

[26] P. Adarsh, P. Rathi, and M. Kumar, "Yolo v3-tiny: Object detection and recognition using one stage improved model," in *2020 6th international conference on advanced computing and communication systems (ICACCS)*. IEEE, 2020, pp. 687–694.

[27] M. D. de Assuncao, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, 2018.

[28] W. Jang, H. Jeong, K. Kang, N. Dutt, and J.-C. Kim, "R-tod: Real-time object detector with minimized end-to-end delay for autonomous driving," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020, pp. 191–204.

[29] B. Cox, J. Galjaard, A. Ghiassi, R. Birke, and L. Y. Chen, "Masa: Responsive multi-dnn inference on the edge," in *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2021, pp. 1–10.

[30] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 68–81.

[31] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 115–127.

[32] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 392–405.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[34] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, "Data security and privacy-preserving in edge computing paradigm: Survey and open issues," *IEEE access*, vol. 6, pp. 18 209–18 237, 2018.

[35] T. Bai, D. Shao, Y. He, S. Fu, and Q. Yang, "P3: A privacy-preserving perception framework for building vehicle-edge perception networks protecting data privacy," in *IEEE International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2023.

[36] T. Bai, S. Fu, and Q. Yang, "Privacy-preserving object detection with secure convolutional neural networks for vehicular edge computing," *Future Internet*, vol. 14, no. 11, p. 316, 2022.

[37] F. Murtagh, "Multilayer perceptrons for classification and regression," *Neurocomputing*, vol. 2, no. 5-6, pp. 183–197, 1991.

[38] B. McWilliams, D. Balduzzi, and J. M. Buhmann, "Correlated random features for fast semi-supervised learning," *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[39] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[40] "Azure private multi-access edge compute (mec) — microsoft azure," https://azure.microsoft.com/en-us/solutions/private-multi-access-edge-compute-mec/overview.

[41] "Aws for the edge – edge computing and storage, 5g, hybrid, iot - amazon web services," https://aws.amazon.com/edge/.

[42] "Edge computing solutions — ibm," https://www.ibm.com/edge-computing.