algorithms

MDPI

Article

Characterizing Perception Deep Learning Algorithms and Applications for Vehicular Edge Computing

Wang Feng [1], Sihai Tang [2], Shengze Wang [3], Ying He [1], Donger Chen [1], Qing Yang [1] and Song Fu [1,*]

[1] Department of Computer Science and Engineering, University of North Texas, Denton, TX 76203, USA; wangfeng@my.unt.edu (W.F.); yinghe@my.unt.edu (Y.H.); dongerchen@my.unt.edu (D.C.); qing.yang@unt.edu (Q.Y.)
[2] Department of Computer Science, Schreiner University, Kerrville, TX 78028, USA; sitang@schreiner.edu
[3] Department of Computer Science and Engineering, University of California, Santa Cruz, CA 95064, USA; shengze@ucsc.edu
* Correspondence: song.fu@unt.edu

Abstract: Vehicular edge computing relies on the computational capabilities of interconnected edge devices to manage incoming requests from vehicles. This offloading process enhances the speed and efficiency of data handling, ultimately boosting the safety, performance, and reliability of connected vehicles. While previous studies have concentrated on processor characteristics, they often overlook the significance of the connecting components. Limited memory and storage resources on edge devices pose challenges, particularly in the context of deep learning, where these limitations can significantly affect performance. The impact of memory contention has not been thoroughly explored, especially regarding perception-based tasks. In our analysis, we identified three distinct behaviors of memory contention, each interacting differently with other resources. Additionally, our investigation of Deep Neural Network (DNN) layers revealed that certain convolutional layers experienced computation time increases exceeding 2849%, while activation layers showed a rise of 1173.34%. Through our characterization efforts, we can model workload behavior on edge devices according to their configuration and the demands of the tasks. This allows us to quantify the effects of memory contention. To our knowledge, this study is the first to *characterize the influence of memory on vehicular edge computational workloads, with a strong emphasis on memory dynamics and DNN layers.*

Keywords: vehicular edge computing; deep learning; autonomous vehicles; perception algorithms

Academic Editor: James Jianqiao Yu

Received: 28 November 2024
Revised: 31 December 2024
Accepted: 3 January 2025
Published: 8 January 2025

Citation: Feng, W.; Tang, S.; Wang, S.; He, Y.; Chen, D.; Yang, Q.; Fu, S. Characterizing Perception Deep Learning Algorithms and Applications for Vehicular Edge Computing. *Algorithms* **2025**, *18*, 31. https://doi.org/10.3390/a18010031

1. Introduction

Perception is crucial for autonomous vehicles (AVs). While AVs have the potential to be highly safe, they are not infallible. All automotive manufacturers prioritize safety and efficiency in their designs, yet unforeseen challenges can still arise. The foremost responsibility of an AV is to ensure the safety of its passengers, making it essential for the vehicle to make critical driving decisions based on its surrounding environment. Additionally, the AV must process data efficiently—from sensing to driving—to respond in a timely manner to both its own actions and those of nearby entities. The inherent nature of AVs presents a gap within traditional edge computing research, and we seek to address some of these challenges below.

In this framework, the primary processing pipeline of an AV involves data flowing from sensors to perception, then to path planning, and finally to actuation. Among these

components, the object detection module is arguably the most vital, as underscored by numerous news reports [1–3]. Object detection demands both speed and accuracy, making Deep Neural Networks (DNNs) the most fitting choice for this module. DNNs have a robust history and are actively researched, finding applications in both academic and industrial settings. Various DNN implementations, such as YOLO, SSD, Faster R-CNN, DETR, Deformable DETR and PointPillars for object detection [4–9], DeepLab, DeepLabV3+, and ViT for image segmentation [10–12], and LaneNet for lane detection [13], demonstrate their effectiveness across tasks.

These DNN workloads are typically executed as edge workloads; however, for a DNN to be effective, it must be continuously trained and optimized to address new challenges that arise over time. For example, the emergence of new climates or unmarked older roads necessitates this adaptability. Consequently, many companies employ human operators and semi-supervised methods to label and train the vast amounts of data generated by vehicles on the road, with each vehicle producing nearly 11 terabytes of data daily [14].

Moreover, while edge computing offers significantly lower latency compared to cloud solutions, it also comes with more limited resources. This scarcity of computing power makes understanding perception workloads on edge devices a critical aspect of vehicle and edge computing [15].

As a mode of transport, AVs must adhere to stringent safety standards, where latency and detection accuracy are paramount [16]. However, meeting real-time requirements is notoriously challenging, as reflected by the extensive research in this domain driven by continuous technological advancements [17]. Given the widespread use of DNNs in perception tasks, the pressing issue is how AVs and edge nodes can effectively manage these workloads without necessitating constant upgrades of sensors and models.

Since most AVs rely on their own data for real-time object detection, they face inherent limitations, particularly regarding sensor blind zones and obstructions [18].

In this paper, we investigate memory contention not only in convolutional neural networks (CNNs) but also across individual layers, focusing on both characterization and prediction. To the best of our knowledge, this work is the first to comprehensively analyze the impact of memory contention on the performance of perception workloads in edge environments. Existing studies often highlight the CPU and GPU as limiting factors, for example, [19] systematically discusses the configuration of CPU and GPU in autonomous driving systems, neglecting the crucial role of memory.

Memory usage is a significant consideration that can profoundly affect the deployment of machine learning models for inference on edge devices. These models typically require substantial memory for training and operation, and their size can escalate rapidly with increased complexity. Consequently, deploying a memory-intensive machine learning model on a resource-constrained edge device can lead to performance issues, such as slower execution times or system crashes. In scenarios where multiple tasks are simultaneously scheduled, only four tasks may be completed despite each having sufficient resources. Therefore, optimizing memory usage is essential for enhancing the performance and reliability of machine learning systems, reducing hardware costs, and improving overall workflow efficiency. Understanding how memory utilization affects workloads at various levels is critical for diverse deployment scenarios.

Our analysis indicates that memory can be effectively generalized through characterization functions to model and predict workload behavior across different machine learning methods and platforms.

By conducting a thorough analysis, we can extract the performance characteristics for each layer within a perception network. We monitor the behavior of individual layers during the inference process under varying constraints and conditions. This information

is vital for understanding model performance and for optimizing its architecture and parameters. However, monitoring the behavior of individual layers can be complex, particularly with intricate models that contain many layers.

Our research reveals that convolutional layers, along with routing, shortcut, and ReLU activation layers, are particularly sensitive to factors such as memory availability. For instance, under constrained memory conditions, some layers may experience significant increases in computation time, with convolutional layers seeing over a 2849% rise, routing layers over 1053%, ReLU layers over 1173.34%, and shortcut layers over 271%. These findings are instrumental in identifying memory bottlenecks in perception networks and developing dynamic memory allocation strategies to enhance performance.

Our main contributions include the following:

- Characterizing and modeling the impact of memory on machine learning inference workloads for edge devices.
- Conducting deep layer analysis to identify specific layers affected by memory contention and over-subsidization.
- Generalizing the edge components that influence machine learning inference workloads and outlining challenges for future research.

The remainder of this paper is structured as follows: Section 2 discusses the background and motivation. Section 3 details the profiling setup and variables. Section 4 presents and analyzes the results. Section 5 elaborates on the findings, and Section 6 concludes the paper.

2. Background and Motivation

As the world increasingly embraces the next generation of sensor-equipped vehicles which has great potential to enhance urban planning and transportation management [20], the supporting infrastructure must evolve accordingly. Modern vehicles, whether fully autonomous or semi-autonomous, demand sophisticated and extensive computational resources to process the data necessary for ensuring passenger safety. To guarantee this safety, computations must be conducted in real-time and be relevant to the vehicle's spatial context. Studies such as [14] explore the future landscape of connected vehicles and clarify the associated challenges, notably the issue of offloading computational tasks away from the vehicle itself.

In recent years, the definition of computing has become less distinct, as many edge devices, like proprietary solutions from Nvidia or even mobile phones, can perform similar tasks. Edge computing finds applications across various domains, including instant diagnosis in healthcare [21], predictive maintenance in manufacturing [22], smart farming in agriculture [23], unmanned aerial vehicles (UAVs) [24], and enhancing Android-based device capabilities [25]. Gradually, the concept of edge Artificial Intelligence [26] emerges. Likewise, edge computing is vital for connected autonomous vehicles (CAVs). Research such as [27] highlights the inefficiencies of onboard computation in CAVs and demonstrates how edge computing can alleviate issues like network congestion and limited processing resources.

Recent studies, including [18,28,29], have investigate and validate the potential of using edge computing for sensor fusion in CAVs. Further works, such as [27,30], examine real-world applications of offloading computations to the edge, revealing a range of possibilities. The discussion around Deep Neural Networks (DNNs) has emerged as a significant focus area in this context as indicated by [14].

Most AVs rely on DNN inferencing for their self-driving capabilities, utilizing state-of-the-art models for object detection, such as YOLO [4], Faster R-CNN [6], Mask R-CNN [31], SSD [5], DETR [7], Deformable DETR [8], and PointPillars [9]. YOLO [4]

pioneers single-shot real-time object detection, balancing speed and accuracy, but struggles with detecting small objects and overlapping objects, as the grid division could lead to imprecise localization in those cases. Faster R-CNN [6] introduces the Region Proposal Network (RPN), improving object detection by generating region proposals. Moreover, Mask R-CNN [31] expands it by adding instance segmentation, allowing both object localization and segmentation. SSD [5] improves single-stage detection by using multi-scale feature maps and default boxes for handling objects at different scales. Both DETR [7] and Deformable DETR [8] try to use and enhance the transformer-based approach to object detection, eliminating region proposals and anchor boxes for a simpler end-to-end pipeline, but both of them are slower than traditional methods like YOLO [4], Faster R-CNN [6], and SSD [5] due to the complexity of the transformer model. PointPillars [9] revolutionizes 3D object detection by applying a CNN to LiDAR point clouds, enabling real-time detection in autonomous driving. While these models perform well, there are trade-offs and optimization opportunities to consider. For instance, to facilitate deployment on lower-end computational hardware, techniques such as quantization, pruning, and architectural modifications are often employed [32–34].

Deploying machine learning models on edge devices, especially those with limited resources, presents several challenges [35]. One significant hurdle is the constrained computing and memory resources available on these devices, which restrict the complexity and size of deployable models. Additionally, power limitations on edge devices further constrain computational energy use [36].

This challenge is particularly pronounced in vehicular edge computing, where most machine learning models require large-weight configurations to function optimally. Thus, models deployed on edge devices must be meticulously optimized for both power and speed, a balance that can be difficult to achieve without compromising accuracy or overall performance.

Another challenge is the lack of standardization in hardware and software platforms, which complicates the development and deployment of machine learning models that maintain consistency across AVs and their associated edge devices. Furthermore, deploying machine learning models on edge devices raises concerns about data privacy and security, as sensitive vehicle data may be processed and stored locally. Addressing these challenges will necessitate advancements in hardware and software optimization, as well as improvements in standardization and data privacy measures [14,37–40].

Traditional improvements in machine learning and neural networks often focus on overall latency per frame and optimizing accuracy through loss function adjustments, typically centering on CPU and GPU impacts while neglecting memory considerations. Although targeted methods like the Pyramid Attention Network [41] and the Region Proposal Network [6] have led to many real-time detection algorithms, these are generally developed without the constraints of edge resources in mind.

Common strategies in the literature for addressing edge memory limitations often involve trade-offs or sacrifices. For example, the authors in [42,43] reduce the full YOLO architecture to decrease computational demands. Similarly, ref. [44] proposes distributing workloads through a pipeline to mitigate hardware constraints like memory contention. Traditional methods, such as [45], consider and attempt to eliminate memory contention for individual tasks.

The contemporary literature addressing memory contention issues for DNNs on edge devices also encounters limitations. Among the various memory-aware middleware approaches, works like MASA [46], along with others such as Deepeye [47], NestDNN [48], and DART [49], tackle CPU and memory issues for low-end edge devices from a middleware perspective, considering the impacts of individual layers on CPU and memory usage.

However, even in the case of MASA, the comprehensive effects of memory are not fully explored. Additionally, due to the methodologies and hardware platforms employed in MASA, the analysis is limited to two-layer types, neglecting single-stage and two-stage foundations. Furthermore, these works do not model memory characteristics as a variable impacting performance; instead, they facilitate DNN processes. Lastly, most modern DNNs leverage PyTorch [50] as their backbone, which is absent from these motivating studies.

Challenges of Characterizing Machine Learning on Edge

Characterizing machine learning workloads on edge devices involves several significant challenges. A primary issue is the wide variety of edge devices and their distinct hardware configurations, making it difficult to generalize performance metrics and operational timings across different systems. Additionally, energy consumption is closely tied to workload intensity and processing duration, further complicating the characterization process. Additionally, the inherent complexity and variability of machine learning models, coupled with the diverse input and network conditions, complicate predictions about model behavior on various edge devices, even when those devices share identical specifications.

Moreover, conducting the detailed profiling of machine learning workloads on edge devices necessitates specialized tools and expertise, which may not be readily accessible to developers or end users. Compounding these challenges is the need to protect client data privacy, as profiling often requires monitoring the processing of sensitive information directly on the device. To address these issues, advancements in standardizing benchmarking tools and developing privacy-preserving methods for edge devices are essential as highlighted in surveys like [51].

Studies such as [52,53] explore the relationship between memory and layer-wise connections to privacy. These papers demonstrate that through layer-wise encoding, it is feasible to encrypt and process data at the layer level with minimal impact on performance for connected autonomous vehicles (CAVs). Similarly, traditional privacy concerns related to memory usage also warrant further investigation.

3. Measuring and Characterizing Perception Workloads on the Edge

Deep learning-based perception networks can be categorized into two main types: single-stage and two-stage object detection. In this study, we focus on both categories, selecting widely recognized representative networks. Specifically, we examine Single-Stage YOLO (both Darknet and PyTorch variants) and Two-Stage Faster R-CNN (PyTorch based). We denote YOLO as ω and Faster R-CNN as θ. By including both single-stage and two-stage methods, we aim to generalize our findings to apply to other methodologies employing similar principles.

Our objective is to explore the distinctive resource demands of each category, particularly regarding memory impacts and the performance bottlenecks that arise from these resource requirements.

Beyond analyzing individual resource impacts, we aim to characterize the workload in a way that allows for predictive modeling. As illustrated in Figure 1, a generalized prediction model for incoming workloads can enhance scheduling on edge nodes. The edge network architecture thus serves as both a communication and computational hub for connected autonomous vehicles (CAVs) utilizing the edge network.

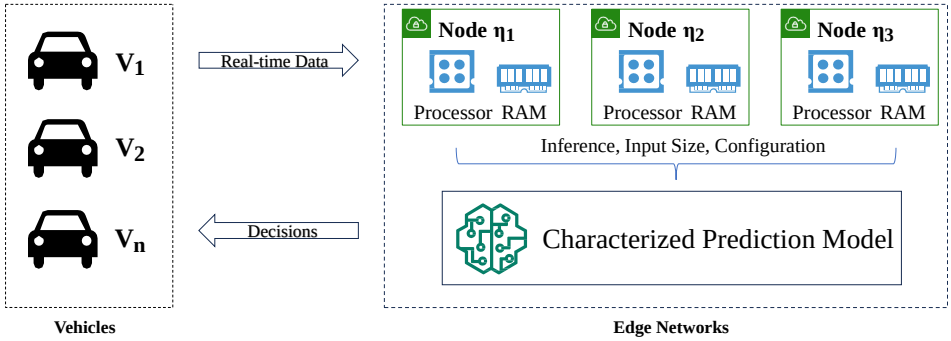

Figure 1. Interaction between autonomous vehicles and vehicular edge computing nodes.

3.1. Profiling Testbed and Setup

To rigorously evaluate the memory contention effects and layer-wise performance, we designed a test environment that closely resembles real vehicular edge scenarios. Our experiments were conducted on two types of edge hardware configurations: (1) a high-end laptop simulating a powerful edge node (Intel Core i7-10750H, Nvidia GeForce RTX 2070, 16 GB DDR4 RAM, 1 TB NVMe SSD), and (2) a Nvidia Jetson Xavier NX as a representative low-end or mid-range edge device. We selected these two platforms to capture a broader spectrum of resource capabilities commonly found in vehicular edge deployments.

Dataset and Workload Parameters: For the object detection tasks, we used inputs derived from standard AV-oriented image sets (e.g., subsets of KITTI or COCO) [54,55]. The input resolutions varied (e.g., 320 × 320, 640 × 480, and 1280 × 720) to emulate camera feeds from different AV sensors. Each resolution set was run multiple times (10 runs per setting) to capture the performance variability and to mitigate any transient system effects (such as GPU thermal throttling or background OS processes). The size of each workload was selected to reflect realistic AV perception tasks, where real-time performance is vital. For each test run, we tracked not only the inference latency but also the peak and average memory consumption, using nvidia-smi, PyTorch's built-in profiler, and Linux-based resource monitoring tools. However, it should be noted that the scope is on system-level profiling rather than perception accuracy metrics. Our approach is agnostic to the dataset, and thus we focus on memory usage and layer execution times behavior, which remain consistent across different datasets.

Simulation Topology: To approximate a distributed edge environment, we modeled each device as if it were an individual edge node receiving inference tasks from one or more autonomous vehicles. Each node could be allocated a certain fraction of tasks, simulating real traffic scenarios where multiple vehicles offload to the same edge. The exact number of concurrent requests varied to provoke different degrees of resource contention. In more constrained tests, we artificially reduced the available RAM or GPU memory to simulate heavy oversubscription.

Reliability and Validity Measures. We repeated each configuration (device × resolution × concurrency level) multiple times to ensure stable averages. We computed standard deviations and confidence intervals (95% CI) to check for statistical consistency. In addition, we systematically logged CPU frequency, GPU utilization, and GPU temperature to verify that the results were not skewed by thermal throttling or one-off system states. Where possible, we cross-checked the measured layer-by-layer timings with independent profiling scripts to validate the internal PyTorch logs. Although our dataset was substantial enough to capture meaningful trends in resource usage, we acknowledge that a larger or more diverse dataset (e.g., a full AV sensor suite) could yield additional insights. This limitation is further discussed in Sections 5 and 6.

3.2. Features and Formulation

To assess how incoming AV workloads will affect available edge nodes, we consider two sets of variables: the inputs from AVs and the resources available from edge nodes. We define the set of AV inputs as $V = \{v_1, v_2, \dots\}$ and the set of edge nodes as $E = \{\eta_1, \eta_2, \dots\}$. Each element $v_i = \{v^i_{\text{data}}, v^i_{\text{task}}\}$ and $\eta_i = \{\eta^i_{\text{conf}}, \eta^i_{\text{queue}}\}$ is monitored and managed by the Prediction Module.

As depicted in Figure 1, upon receiving input requests from vehicles, the edge nodes within the service range can process these tasks. Each edge node evaluates the incoming workload requests and determines its capacity to handle them. The nodes then communicate their task requirements and resource availability to the edge manager, which efficiently allocates the workload to an edge node capable of completing the task within an acceptable timeframe.

To formulate potential edge configurations, we define the following key variables: processor resource, RAM as the memory resource, workload size (derived from the requested service and input size), and time (as the quantifying metric).

4. Analysis and Key Insights

The variability in edge deployment hardware and task types presents a significant challenge. Diverse AV workload requests and differences in hardware performance on edge nodes complicate planning, ultimately affecting the reliability of DNN inferencing and presenting a fundamental challenge for robust workload characterization.

4.1. Memory and Computation Load

As discussed in Section 2, memory contention is a critical factor when latency in time or performance is essential. Figure 2 illustrates two distinct states, emphasizing the necessity of managing memory contention effectively in these scenarios.

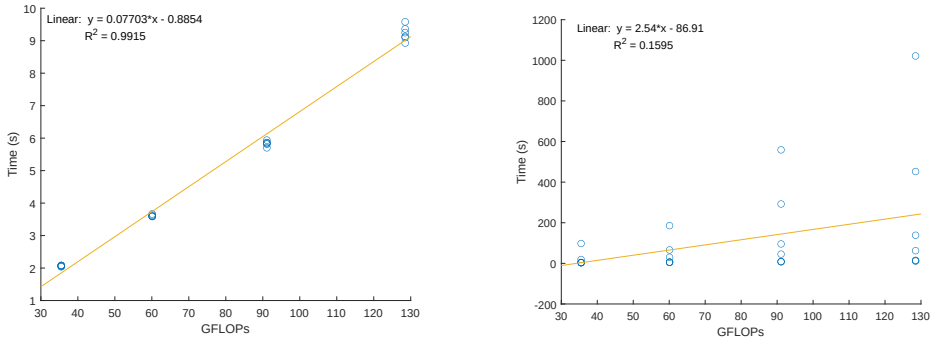

Figure 2. (**Left**) Single-stage perception deep learning application with and (**Right**) without memory resource contention.

In our profiling, we identify three scenarios regarding memory resources:

- Scenario 1: No memory resource contention.
- Scenario 2: Memory resource contention typically encountered under heavy workloads.
- Scenario 3: Extreme lack of memory resources.

In Scenario 1, memory resources are abundant for all tasks, with no bottlenecks. In Scenario 2, however, memory resources are strained due to access contention or limited availability. Notably, temperature and system bus bottlenecks are not considered in Scenario 2, as these conditions may lead to system instability. Scenario 3 accounts for severe memory allocation constraints, from low to minimal operational limits.

In Scenario 1 (Figure 2(left)), the model remains consistent with a CPU-dominated linear model, which we can express as

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

where $\gamma = GFLOPs$, representing the computational load required by the workload.

However, when applying Equation (1) to the same set of workloads under memory limitations, the model no longer holds true as depicted in Figure 2(right). Traditional optimization and scheduling methods may alleviate some issues, but they cannot fully resolve the challenges posed by memory contention.

Thus, to accurately characterize the impact of memory contention across various platforms, further characterization is necessary for workloads in Scenarios 2 and 3, which encounter moderate to heavy memory contention.

By modeling the behavior of the algorithms under varying resource constraints, we can profile and characterize the effects of memory contention, with Faster R-CNN denoted as (θ) and YOLO as (ω). Initially, we profile the maximum amount of RAM needed for each machine learning method, represented as θ_{MAX} for Faster R-CNN and ω_{MAX} for YOLO. Each ML method is tested with progressively less available RAM until task failure occurs.

As shown in Figure 3, we profile the workload characteristics under Scenario 3 and observe a clear correlation with the amount of memory available. From the data collected, we can model the behaviors of the two ML workloads using Equations (2) and (3), respectively. Here, the CPU frequency is represented as ν = CPU Power, and memory is denoted as β = Memory. It is important to note that while an R^2 value of at least 0.9 is deemed acceptable, extreme cases, such as operating ω with 100 MB or less, exhibit higher variance when predicted using this model.

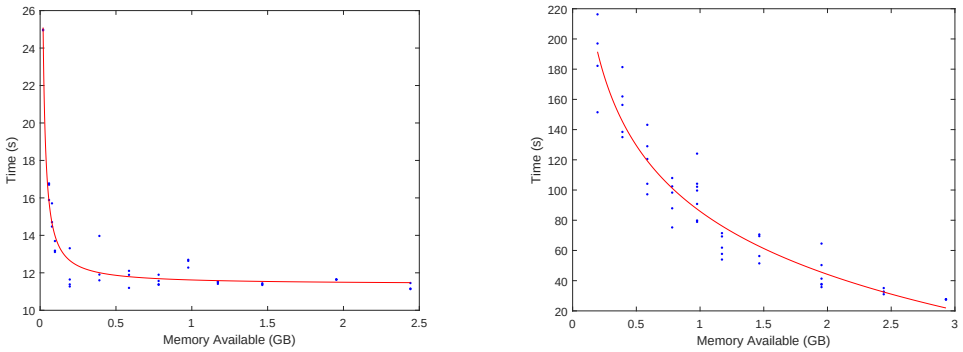

Figure 3. Performance of perception deep learning applications single-stage (**Left**) and two-stage (**Right**) on high-end vehicular edge devices with memory contention.

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

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

To ensure consistency in edge workload parameters for Scenario 3 across different platforms, we analyze the same scenario on the Jetson device, as shown in Figure 4. While the Jetson device exhibits less variance in the data, its behavior aligns closely with our previous findings. The second set of models derived for Faster R-CNN (θ), and YOLO (ω) is expressed in Equations (4) and (5).

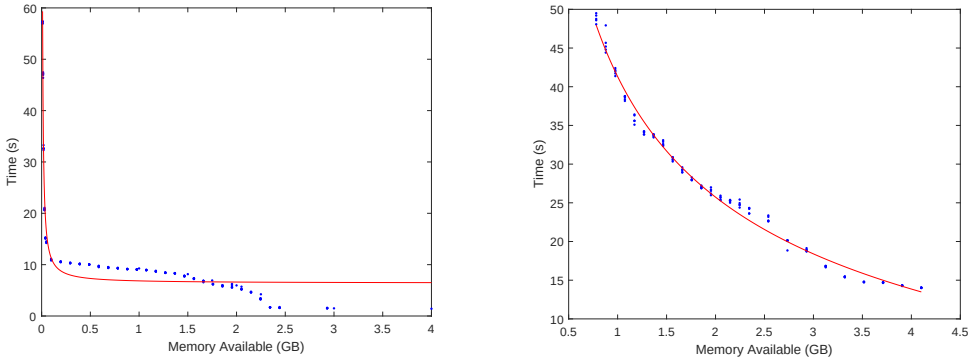

Figure 4. Performance of perception deep learning applications single-stage (**Left**) and two-stage (**Right**) on low-end vehicular edge devices with memory contention.

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

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

With the Jetson Memory model, we achieve an $R^2 = 0.98$ for θ and $R^2 = 0.87$ for ω.

This characterization is beneficial for industrial cost estimation, providing insights into how many incoming tasks an edge node can manage. With the impact of memory clearly defined, we now turn our attention to how it affects each individual layer of the models.

4.2. Layer-Wise Characterization

Building on the insights gained from memory profiling, we can delve into the layer-specific analysis of our machine learning methods. Both YOLO and Faster R-CNN utilize several common layers, including convolutional layers, activation layers, and operations such as max-pooling. Figure 5 illustrates the differences in layer performance under normal operation versus resource-constrained conditions, highlighting how memory contention disproportionately affects specific layers within the architecture.

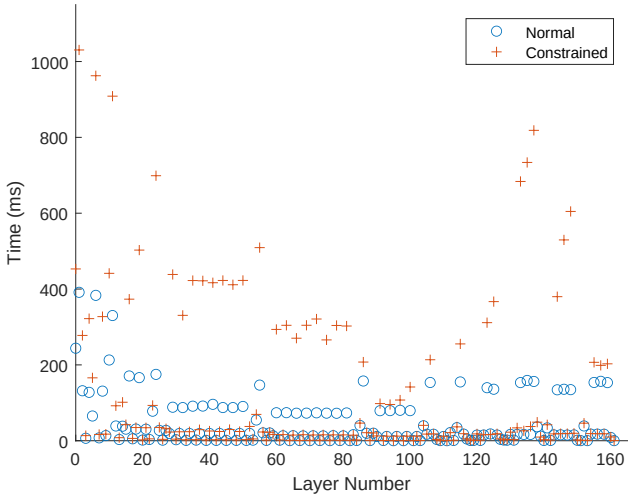

Figure 5. Fine-grained layer-wise performance of single-stage perception DL application for both no contention and under/oversubsidized contention.

Given the shared building blocks, we conduct a layer-by-layer workload profiling for both scenarios. The layers that experience significant impacts are depicted in Figure 6 for YOLO and Figure 7 for Faster R-CNN. A detailed analysis of the gathered data reveals that the layers that are most affected are primarily located at the beginning of the architecture

for both models, ω and θ. Table 1 summarizes the layers that are most impacted in YOLO, while Table 2 details the affected layers in Faster R-CNN.

Table 1. Performance of single-stage perception deep learning network layers with and without memory contention.

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

Table 2. Performance of two-stage perception deep learning network layers with and without memory contention.

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

Starting with Figure 6(left), we observe that two layers—Layer 1 and Layer 6—are significantly impacted by memory contention. Both layers are convolutional layers with a theoretical computational load of 3.407 GFLOPs. Despite this seemingly substantial load, similar loads are distributed across various other layers in the architecture. Notably, Layer 1 exhibits a remarkable 27.65 times increase in execution time, while Layer 6 demonstrates a 29.49 times increase compared to the non-contention scenario. These findings provide valuable insights into the areas most affected by resource constraints within the YOLO architecture, highlighting the critical layers that may benefit from targeted optimization efforts.

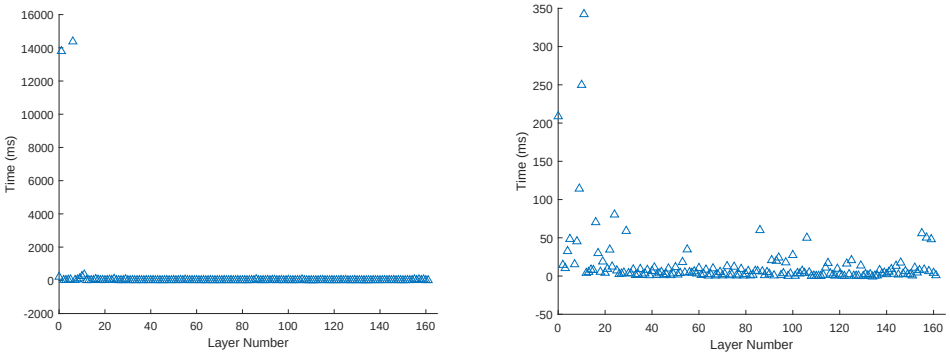

Figure 6. Impacts of memory contention on single-stage perception DL network layer on vehicular edge.

However, in Figure 6(right), after removing the two heavily impacted layers from Figure 6(left), we identify other layers that also show notable increases in latency. In our profiling results, Layers 9 and 20 reveal an unusual degree of increase when compared to both their theoretical GFLOP values and the non-contention latency. Specifically, Layer 9, which functions as a routing layer with a near-zero GFLOP, experiences a staggering latency

increase of over 1053%. Similarly, Layer 20, a shortcut layer with an almost negligible GFLOP of 0.001, demonstrates a latency increase of 271%.

While it is expected that the initial convolutional layers would be heavily affected by memory constraints—primarily due to the necessity of loading input data from storage—the significant impacts observed in Layers 9 and 20 suggest intriguing avenues for future research.

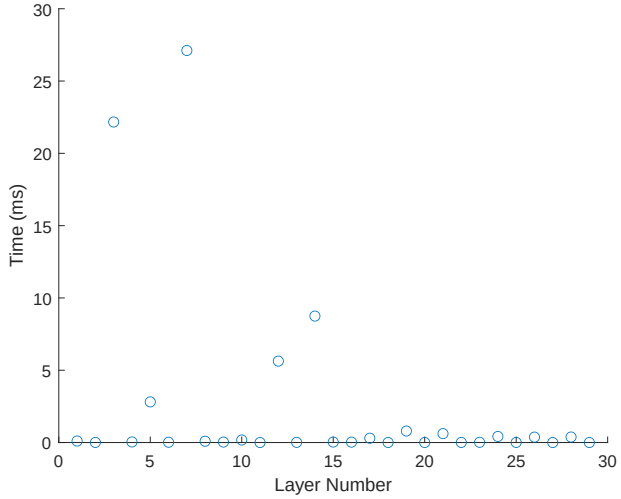

Figure 7. Impacts of memory contention on two-stage perception DL network layers.

Next, we analyze the layer data for Faster R-CNN as shown in Figure 7. Here, five layers are identified as most affected, with Layers 3 and 7 standing out prominently. Layer 3, a ReLU activation layer, exhibits a dramatic increase of 1173%, while Layer 7, a convolutional layer, shows an even more severe increase of 2339.77%.

4.3. Compute Power and Processor Characterization

On both of our platforms, we began with a consistent input load for the two machine learning methods. Figure 8 illustrates the performance of Faster R-CNN across four distinct CPU power Thermal Design Power (TDP) and core configurations, using inference time as the key metric to characterize the workload. Based on the data points collected in Figure 8, right, we can model the impact of Scenario 1 for Faster R-CNN as follows, where $\nu = CPU$ and frequency is in gigahertz.

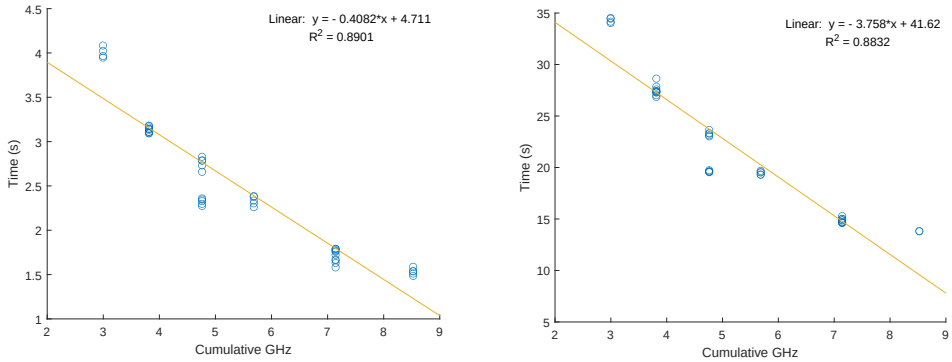

Figure 8. Performance of perception deep learning applications on Jetson with varying CPU configurations. (**Left**) Single-stage perception network. (**Right**) Two-stage perception network.

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

Similarly, Figure 8(left) presents the profiling results for YOLO under the same CPU power and core configurations. We can also formulate the model for the impacts of Scenario 1 on the YOLO workload as follows:

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

Incorporating the memory variable from Scenario 2, we arrive at Equation (8) for Faster R-CNN, with memory expressed in gigabytes and frequency in gigahertz:

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

Likewise, when characterizing the complete workload, Scenario 2 for YOLO is represented by Equation (9), which can be expressed as follows:

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

4.4. Workload Size and Resource Requirement

The CPU profiling reveals a direct linear relationship between the CPU computational power and inference performance for both models as illustrated in Figure 8.

But the input size of the task also influences the amount of computational power required. For instance, processing higher-resolution sensor data demands significantly more computational power compared to low-resolution data. To explore this further, we perform profiling with various input sizes as depicted in Figure 9.

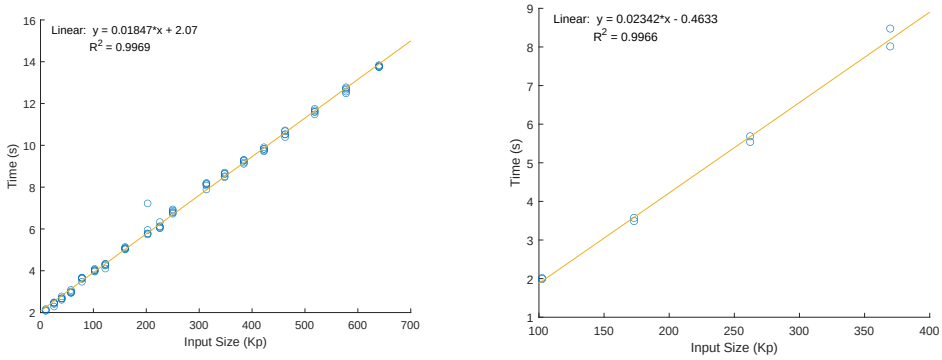

Figure 9. Impacts of edge input for single-stage (**Left**) and two-stage (**Right**) perception DL applications.

However, when we introduce an additional variable, memory, into the equation, the profiling results change significantly as illustrated in Figure 2. Applying the direct linear formula with adjusted constants to Figure 2(right) shows that the R^2 fit for the formulation based solely on the CPU variable becomes increasingly inaccurate.

4.5. Insight Analysis and Generalization

Using the data from our profiling to identify the variables that influence ML workloads, we can now work on generalizing our prediction model.

Drawing from the key insights in Formulas (2)–(5), we can summarize the findings for Scenario 3 as follows:

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

where k_i and a_i represent the variable coefficient constants associated with each edge node, while Q_i denotes the respective constant value, and β indicates the currently available memory. To accurately represent Scenario 3, the variable ν is removed from Equation (10). We can account for the variance in thermal limitations using the rolling window average in Equation (11), where n marks a specific point in time, enhancing accuracy. The rolling window for k_n and a_n can be defined such that $k_1 = k_2 \ldots k_n = k_{n+1}$, with $n+1$ representing the current time constant for Equation (10):

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

By integrating the two characterization models, we can formulate a decision tree for the task queue c running on the edge. Here, $\beta_i^{threshold}$ for task i signifies the memory threshold for Scenario 3, where the influence of the CPU variable is overshadowed by the effects of memory. This threshold varies depending on the ML algorithm; in our results, it is approximately 100 MB for YOLO and 800 MB for Faster R-CNN. The decision tree for predicting workload is outlined as follows:

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

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

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

Based on the profiling results, the first key insight highlights the effects of limited memory resources on a given workload. Traditional CPU-based scheduling approaches fail to consider physical memory constraints, which can lead to issues like the over-subsidization of an edge node. Furthermore, the amount of data needed for a straightforward predictive model derived from our insights is quite small, enabling quick and efficient customization.

We identify the characteristics of various edge node components relevant to CAV-based inference workloads, categorizing the variables for CPU and input size as linear while implementing specific measures to address over-subsidization scenarios.

Further profiling of layer performance reveals that only a few specific layers are impacted by memory contention. As shown in Figures 6 and 7, we identify the affected layers for ω and θ, respectively. These findings facilitate a more targeted approach to researching and optimizing ML methods for edge devices, particularly in scenarios involving memory contention or lower hardware specifications.

5. Implications and Discussion

In our generalization, we initially explored a non-linear approach to account for all the variables discussed in Section 4 simultaneously. To achieve this, we tested various modeling methods. The results of including all our targeted variables are presented in Table 3. While

Linear Regression provides a reasonable level of model accuracy, it is outperformed by methods such as MLP [56] and Correlated Nystrom Views [57].

Table 3. Traditional and ML predictor models based on the features listed in Table 1. Coefficients of layers point to Equation (1).

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

Through additional testing, we attempted to train and fine-tune the aforementioned models using actual data, comparing their predictive capabilities across different platforms. However, due to the limited sample size, the models encountered overfitting issues. To address this, we applied SMOTE [58] to upsample our data where necessary, aiming for better generalization. Unfortunately, this process proved to be cumbersome and requires further optimization through research.

While we are confident that the ML modeling approach based on the identified variables will outperform the generalized formulations presented, it will necessitate a larger dataset for effectiveness and will require retraining for each individual edge node in a self-updating loop, which will demand additional resources.

5.1. Unusual Findings

Several unusual findings emerged during our research that warrant further attention. Firstly, our profiling revealed that oversubsidized workloads trigger the Out-Of-Memory (OOM) killer in Nvidia Jetson but not on other platforms using the same Linux kernel and OOM policy. Since this issue exceeds the scope of this paper, we did not explore it further, but it presents a potential avenue for optimizing CAV workload scheduling.

Another anomaly noted during profiling was the discrepancy between the theoretical computational load and the actual latency observed in the workload. Specifically, as discussed in the layer-wise profiling section, we found that shortcut layers and routing layers exhibited significantly higher latency than standard convolutional layers. While this could be attributed to insufficient memory, it does not fully account for the substantial increase in latency we measured. Additionally, both shortcut and routing layers have a minimal computational footprint, making this finding particularly relevant for optimization research. Once again, due to the technical complexities involved, we did not investigate this phenomenon further. We hypothesize that this latency increase may stem from a linear or sequential architecture workflow. Future research could explore base ML algorithms and those that can be parallelized, such as transformers, to determine if a method can be developed to improve the consistency between the theoretical performance and real-world outcomes.

Although these findings did not impact the workload profiling directly, they highlight additional opportunities for future investigation.

5.2. Limitations

There are several limitations that are noteworthy and go beyond the scope of a single paper. However, we believe that addressing these will be great future work directions to navigate.

First is the partial consideration of AV dynamics. This paper zeroes in on memory contention and latency in deep learning workloads but does not explicitly account for vehicular mobility models or dynamic network conditions (e.g., handoff between edge nodes as a vehicle moves). While we believe the memory-contention insights are broadly applicable, real-time vehicular scenarios might involve intermittent connectivity or variable network latency that further affects resource allocation.

Second is the limitations of open source datasets. It is a well-known fact that these publicly available datasets are just a very small representation of the conditions faced in real life. Any models created or studied within the scope of these publicly available datasets will run into the same issues of being narrow in scope and inapplicable to real-life needs. We believe that more diverse datasets are needed to refine the observations made in this paper.

Third, we also see that certain layers (e.g., routing and shortcut layers in YOLO) exhibited disproportionate latency spikes under memory contention. While we hypothesize that these anomalies stem from architecture design and memory-access bottlenecks, our analysis does not fully isolate all possible causes (e.g., overhead from framework-level memory allocation, PCIe contention, or caching effects). A deeper hardware or machine-level study could offer more conclusive explanations.

Finally, the current work does not quantitatively evaluate how encryption or obfuscation techniques impact memory contention. In safety-critical vehicular settings, ensuring data confidentiality might introduce extra overhead, which we have not yet modeled.

5.3. Further Explorations

A promising avenue for exploration based on our findings is the development of accurate and efficient profiling tools and methodologies that can manage the complexity and variability of machine learning models and edge device hardware. A comprehensive understanding of workload behavior can lead to the establishment of standardized benchmarks and metrics targeting variables for more generalized use cases.

Furthermore, our findings indicate a need to address privacy considerations. While our profiling does not incorporate privacy algorithms and methods applicable to the selected workloads, the integration of privacy-preserving techniques and data obfuscation methods may reveal different workload characteristics and provide additional insights. Considering privacy in workload profiling could offer a more complete understanding of the challenges and opportunities at hand.

6. Conclusions and Future Works

Machine learning and AI workloads have become stable components of our current society, driven by the widespread adoption of edge technology by major companies such as Microsoft Azure, Amazon AWS, and IBM [59–61]. However, utilizing on-road edge to offload workloads from connected and autonomous vehicles (CAVs) introduces many uncertainties. Existing research on optimizing machine learning for edge devices lacks the in-depth insights necessary for edge users and device providers to accurately predict the behavior of machine learning workloads across various configurations. In this paper, we tackle these challenges and identify areas for potential improvement. We propose a novel approach to optimizing workloads by considering edge device parameters, workload

inputs, and algorithm architectures. The profiling and insights provided in our approach can be generalized beyond the specific machine learning algorithms discussed here.

While our models are fundamental, they provide a clearer characterization of workload behavior compared to traditional machine learning (ML) models. Although some models, like MLP and Correlated Nystrom Views, can predict workload behavior, they require extensive training and struggle to accommodate diverse ML inference algorithms.

By characterizing and generalizing our findings, we offer valuable insights into the performance and capabilities of edge devices for machine learning workloads. Our analysis indicates that even under memory contention, algorithms such as YOLO and Faster R-CNN can successfully complete their tasks, suggesting that resource oversubscription on edge devices is a feasible option for non-time-sensitive applications.

Moreover, utilizing memory and layer-wise information allows for the optimization of both the architecture and deployment of machine learning models, enhancing the efficiency of edge devices and reducing energy consumption. Our analysis identifies critical bottlenecks in machine learning workloads as detailed in Sections 4 and 5.

The ability to predict the execution time of specific layers presents several advantages for future research. First, it enables the optimization of model architecture and hyperparameters by pinpointing the most time-consuming layers and enhancing their performance. Second, it improves model efficiency during training or inference by scheduling layers to maximize available resources and minimize overall execution time. Finally, it supports the development of new algorithms and techniques that consider the performance characteristics of individual layers, leading to the creation of more efficient and accurate machine learning models.

The contributions of our paper are significant. Our proposed approach offers a targeted method for predicting the performance and maximum workload capabilities of edge devices for machine learning inference tasks. These insights empower other researchers and service providers to pursue future research on more efficient algorithms and resource management strategies, including memory and processing power, ultimately reducing contention and enhancing overall performance. Additionally, exploring distributed systems and edge-to-cloud architectures could help mitigate resource contention and improve scalability.

In conclusion, there are numerous potential areas for future research in profiling machine learning workloads on edge devices. Addressing these challenges will be vital for facilitating the widespread deployment of machine learning models in edge environments. By investigating these areas, researchers can advance efficient algorithms, resource management strategies, profiling tools, and privacy-preserving techniques, ultimately enhancing the seamless integration of machine learning within edge computing frameworks.

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

Funding: 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.

Data Availability Statement: Dataset available on request from the authors.

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

References

1. Exclusive: Surveillance Footage of Tesla Crash on Bay Bridge. Available online: https://theintercept.com/2023/01/10/tesla-crash-footage-autopilot/ (accessed on 28 November 2024).
2. 93% Have Concerns About Self-Driving Cars According to New Forbes Legal Survey. Available online: https://www.forbes.com/advisor/legal/auto-accident/perception-of-self-driving-cars/ (accessed on 18 November 2024).
3. Automated Vehicles for Safety. Available online: https://www.nhtsa.gov/vehicle-safety/automated-vehicles-safety (accessed on 18 November 2024).
4. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
5. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
6. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 1137–1149. [CrossRef] [PubMed]
7. Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. End-to-end object detection with transformers. In Proceedings of the European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 213–229.
8. Zhu, X.; Su, W.; Lu, L.; Li, B.; Wang, X.; Dai, J. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv* **2020**, arXiv:2010.04159.
9. Lang, A.H.; Vora, S.; Caesar, H.; Zhou, L.; Yang, J.; Beijbom, O. Pointpillars: Fast encoders for object detection from point clouds. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 12697–12705.
10. Chen, L.C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A.L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *40*, 834–848. [CrossRef]
11. Chen, L.C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 801–818.
12. Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929.
13. Neven, D.; De Brabandere, B.; Georgoulis, S.; Proesmans, M.; Van Gool, L. Towards end-to-end lane detection: An instance segmentation approach. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Suzhou, China, 26–30 June 2018; pp. 286–291.
14. Lu, S.; Shi, W. Vehicle computing: Vision and challenges. *J. Inf. Intell.* **2022**, *1*, 23–35. [CrossRef]
15. Waheed, A.; Shah, M.A.; Mohsin, S.M.; Khan, A.; Maple, C.; Aslam, S.; Shamshirband, S. A comprehensive review of computing paradigms, enabling computation offloading and task execution in vehicular networks. *IEEE Access* **2022**, *10*, 3580–3600. [CrossRef]
16. Liu, L.; Dong, Z.; Wang, Y.; Shi, W. Prophet: Realizing a Predictable Real-time Perception Pipeline for Autonomous Vehicles. In Proceedings of the 2022 IEEE Real-Time Systems Symposium (RTSS), Houston, TX, USA, 5–8 December 2022; pp. 305–317.
17. Liu, L.; Zhao, M.; Yu, M.; Jan, M.A.; Lan, D.; Taherkordi, A. Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 2169–2182. [CrossRef]
18. Chen, Q.; Ma, X.; Tang, S.; Guo, J.; Yang, Q.; Fu, S. F-cooper: Feature based cooperative perception for autonomous vehicle edge computing system using 3D point clouds. In Proceedings of the ACM/IEEE Symposium on Edge Computing (SEC), Arlington, WV, USA, 7–9 November 2019.
19. Wang, X.; Maleki, M.A.; Azhar, M.W.; Trancoso, P. Moving Forward: A Review of Autonomous Driving Software and Hardware Systems. *arXiv* **2024**, arXiv:2411.10291.
20. Yu, C.; Xie, X.; Huang, Y.; Qiu, C. Harnessing llms for cross-city od flow prediction. In Proceedings of the 32nd ACM International Conference on Advances in Geographic Information Systems, Atlanta, GA, USA, 29 October 2024; pp. 384–395.
21. Boda, V.V.R. Edge Computing in Healthcare: What It Is and Why It Matters. *MZ Comput. J.* **2024**, *5*, 1–18.
22. Sharma, M.; Tomar, A.; Hazra, A. Edge computing for industry 5.0: Fundamental, applications and research challenges. *IEEE Internet Things J.* **2024**, *11*, 19070–19093. [CrossRef]
23. Sathya, D.; Thangamani, R.; Balaji, B.S. The Revolution of Edge Computing in Smart Farming. In *Intelligent Robots and Drones for Precision Agriculture*; Springer: Berlin/Heidelberg, Germany, 2024; pp. 351–389.
24. Sun, H.; Zhang, B.; Zhang, X.; Yu, Y.; Sha, K.; Shi, W. FlexEdge: Dynamic Task Scheduling for a UAV-Based On-Demand Mobile Edge Server. *IEEE Internet Things J.* **2022**, *9*, 15983–16005. [CrossRef]
25. Yao, Y.; Liu, B.; Zhao, Y.; Shi, W. Towards Edge-enabled Distributed Computing Framework for Heterogeneous Android-based Devices. In Proceedings of the 2022 IEEE/ACM 7th Symposium on Edge Computing (SEC), Seattle, WA, USA, 5–8 December 2022; pp. 531–536. [CrossRef]

26. Gill, S.S.; Golec, M.; Hu, J.; Xu, M.; Du, J.; Wu, H.; Walia, G.K.; Murugesan, S.S.; Ali, B.; Kumar, M.; et al. Edge AI: A taxonomy, systematic review and future directions. *Clust. Comput.* **2025**, *28*, 18. [CrossRef]
27. Tang, S.; Chen, B.; Iwen, H.; Hirsch, J.; Fu, S.; Yang, Q.; Palacharla, P.; Wang, N.; Wang, X.; Shi, W. Vecframe: A vehicular edge computing framework for connected autonomous vehicles. In Proceedings of the 2021 IEEE International Conference on Edge Computing (EDGE), Chicago, IL, USA, 5–10 September 2021; pp. 68–77.
28. Luo, Q.; Li, C.; Luan, T.H.; Shi, W. Collaborative Data Scheduling for Vehicular Edge Computing via Deep Reinforcement Learning. *IEEE Internet Things J.* **2020**, *7*, 9637–9650. [CrossRef]
29. Luo, Q.; Hu, S.; Li, C.; Li, G.; Shi, W. Resource scheduling in edge computing: A survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2131–2165. [CrossRef]
30. Hu, S.; Li, G.; Shi, W. LARS: A Latency-Aware and Real-Time Scheduling Framework for Edge-Enabled Internet of Vehicles. *IEEE Trans. Serv. Comput.* **2023**, *16*, 398–411. [CrossRef]
31. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2961–2969.
32. Cai, Y.; Hua, W.; Chen, H.; Suh, G.E.; De Sa, C.; Zhang, Z. Structured pruning is all you need for pruning CNNs at initialization. *arXiv* **2022**, arXiv:2203.02549.
33. Hou, Z.; Qin, M.; Sun, F.; Ma, X.; Yuan, K.; Xu, Y.; Chen, Y.K.; Jin, R.; Xie, Y.; Kung, S.Y. Chex: Channel exploration for CNN model compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 12287–12298.
34. Cao, S.; Ma, L.; Xiao, W.; Zhang, C.; Liu, Y.; Zhang, L.; Nie, L.; Yang, Z. 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, Long Beach, CA, USA, 15–20 June 2019; pp. 11216–11225.
35. Choi, P.; Kim, J.; Kwak, J. Impact of Joint Heat and Memory Constraints of Mobile Device in Edge-Assisted On-Device Artificial Intelligence. In Proceedings of the 2nd International Workshop on Networked AI Systems, Ternopil, Ukraine, 12–14 June 2024; pp. 31–36.
36. Mavromatis, I.; Katsaros, K.; Khan, A. Computing Within Limits: An Empirical Study of Energy Consumption in ML Training and Inference. *arXiv* **2024**, arXiv:2406.14328.
37. Parkinson, S.; Ward, P.; Wilson, K.; Miller, J. Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Trans. Intell. Transp. Syst.* **2017**, *18*, 2898–2915. [CrossRef]
38. He, Q.; Cui, G.; Zhang, X.; Chen, F.; Deng, S.; Jin, H.; Li, Y.; Yang, Y. A game-theoretical approach for user allocation in edge computing environment. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *31*, 515–529. [CrossRef]
39. Zhang, Z.; Fu, S. Characterizing Power and Energy Usage in Cloud Computing Systems. In Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Athens, Greece, 29 November–1 December 2011. [CrossRef]
40. Guan, Q.; Zhang, Z.; Fu, S. Ensemble of Bayesian Predictors for Autonomic Failure Management in Cloud Computing. In Proceedings of the IEEE International Conference on Computer Communications and Networks, Maui, HI, USA, 31 July–4 August 2011. [CrossRef]
41. Li, H.; Xiong, P.; An, J.; Wang, L. Pyramid attention network for semantic segmentation. *arXiv* **2018**, arXiv:1805.10180.
42. Carrasco, D.P.; Rashwan, H.A.; García, M.Á.; Puig, D. T-YOLO: Tiny vehicle detection based on YOLO and multi-scale convolutional neural networks. *IEEE Access* **2021**, *11*, 22430–22440. [CrossRef]
43. Adarsh, P.; Rathi, P.; Kumar, M. YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In Proceedings of the 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 6–7 March 2020; pp. 687–694.
44. de Assuncao, M.D.; da Silva Veith, A.; Buyya, R. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *J. Netw. Comput. Appl.* **2018**, *103*, 1–17. [CrossRef]
45. Jang, W.; Jeong, H.; Kang, K.; Dutt, N.; Kim, J.C. R-TOD: Real-time object detector with minimized end-to-end delay for autonomous driving. In Proceedings of the 2020 IEEE Real-Time Systems Symposium (RTSS), Houston, TX, USA, 1–4 December 2020; pp. 191–204.
46. Cox, B.; Galjaard, J.; Ghiassi, A.; Birke, R.; Chen, L.Y. Masa: Responsive Multi-DNN Inference on the Edge. In Proceedings of the 2021 IEEE International Conference on Pervasive Computing and Communications (PerCom), Kassel, Germany, 22–26 March 2021; pp. 1–10. [CrossRef]
47. Mathur, A.; Lane, N.D.; Bhattacharya, S.; Boran, A.; Forlivesi, C.; Kawsar, F. 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, Niagara Falls, NY, USA, 16 June 2017; pp. 68–81.

48. Fang, B.; Zeng, X.; Zhang, M. 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, New Delhi, India, 29 October–2 November 2018; pp. 115–127.
49. Xiang, Y.; Kim, H. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In Proceedings of the 2019 IEEE Real-Time Systems Symposium (RTSS), Hong Kong, China, 3–6 December 2019; pp. 392–405.
50. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 1–12.
51. Zhang, J.; Chen, B.; Zhao, Y.; Cheng, X.; Hu, F. Data security and privacy-preserving in edge computing paradigm: Survey and open issues. *IEEE Access* **2018**, *6*, 18209–18237. [CrossRef]
52. Bai, T.; Shao, D.; He, Y.; Fu, S.; Yang, Q. P3: A Privacy-Preserving Perception Framework for Building Vehicle-Edge Perception Networks Protecting Data Privacy. In Proceedings of the IEEE International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 24–27 July 2023.
53. Bai, T.; Fu, S.; Yang, Q. Privacy-Preserving Object Detection with Secure Convolutional Neural Networks for Vehicular Edge Computing. *Future Internet* **2022**, *14*, 316. [CrossRef]
54. COCO—Common Objects in Context. Available online: http://cocodataset.org/#download (accessed on 10 January 2020).
55. Geiger, A.; Lenz, P.; Urtasun, R. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR), Providence, RI, USA, 16–21 June 2012.
56. Murtagh, F. Multilayer perceptrons for classification and regression. *Neurocomputing* **1991**, *2*, 183–197. [CrossRef]
57. McWilliams, B.; Balduzzi, D.; Buhmann, J.M. Correlated random features for fast semi-supervised learning. *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 1–9.
58. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [CrossRef]
59. Azure Private Multi-Access Edge Compute (MEC) | Microsoft Azure. Available online: https://azure.microsoft.com/en-us/solutions/private-multi-access-edge-compute-mec/#overview (accessed on 28 November 2024).
60. AWS for the Edge—Edge Computing and Storage, 5G, Hybrid, IoT—Amazon Web Services. Available online: https://aws.amazon.com/edge/ (accessed on 28 November 2024).
61. Edge Computing Solutions | IBM. Available online: https://www.ibm.com/edge-computing (accessed on 28 November 2024).