VALO: A Versatile Anytime Framework for LiDAR-Based Object Detection Deep Neural Networks

Ahmet Soyyigit, Shuochao Yao, and Heechul Yun, *Senior Member, IEEE*

***Abstract*—This work addresses the challenge of adapting dynamic deadline requirements for the LiDAR object detection deep neural networks (DNNs). The computing latency of object detection is critically important to ensure safe and efficient navigation. However, the state-of-the-art LiDAR object detection DNNs often exhibit significant latency, hindering their real-time performance on the resource-constrained edge platforms. Therefore, a tradeoff between the detection accuracy and latency should be dynamically managed at runtime to achieve the optimum results. In this article, we introduce versatile anytime algorithm for the LiDAR Object detection (VALO), a novel data-centric approach that enables anytime computing of 3-D LiDAR object detection DNNs. VALO employs a deadline-aware scheduler to selectively process the input regions, making execution time and accuracy tradeoffs without architectural modifications. Additionally, it leverages efficient forecasting of the past detection results to mitigate possible loss of accuracy due to partial processing of input. Finally, it utilizes a novel input reduction technique within its detection heads to significantly accelerate the execution without sacrificing accuracy. We implement VALO on the state-of-the-art 3-D LiDAR object detection networks, namely CenterPoint and VoxelNext, and demonstrate its dynamic adaptability to a wide range of time constraints while achieving higher accuracy than the prior state-of-the-art. Code is available at https://github.com/CSL-KU/VALOgithub.com/CSL-KU/VALO.**

***Index Terms*—3-D object detection, anytime computing, LiDAR.**

I. Introduction

PERCEPTION plays a vital role in autonomous vehicles. Its primary objective is to identify and categorize objects of interest (e.g., cars and pedestrians) within the operational environment. While humans excel at this task effortlessly, it presents a significant challenge for the computers. For the object detection in 3-D space, LiDAR-based object detection deep neural networks (DNNs) [1], [2], [3] have emerged as an effective approach as they can provide highly accurate position, orientation, size, and velocity estimates.

In autonomous vehicles, however, the object detection results must not only be accurate but also timely as the outdated results are of little use in the path planning of a fast-moving autonomous vehicle. Unfortunately, the LiDAR object detection DNNs are often computationally expensive and thus exhibit significant latency, especially when running on resource-constrained embedded computing platforms. Moreover, they lack the ability to dynamically trade execution time and accuracy, which makes it difficult to adapt to dynamically changing real-time requirements in autonomous vehicles [4], [5]. For example, when a vehicle moves at a high speed, fast detection may be more important than high accuracy (e.g., correct object classification) in order to avoid collision in a timely manner. On the other hand, when the vehicle moves slowly in a complex urban environment, accurate detection may be more important than the fast detection for safe navigation.

To enable schedulable tradeoffs between the accuracy and latency in perception, the prior research efforts have focused on the vision-based DNNs [6], [7], [8], [9], [10]. Model-level innovations, such as early exit architectures [9] have been widely adopted, where these models incorporate additional output layers at the intermediate stages, allowing the network to make predictions before the full depth of the model is utilized. Nonetheless, these enhancements come with a tradeoff. The repeated activation of the intermediate output layers at several phases leads to a significant increase in the computational overhead. This issue is particularly pronounced in applications requiring complex detection heads capable of producing granular object-level predictions, such as LiDAR-based object detection and segmentation tasks. Recently, AnytimeLidar [11] introduced a capability to bypass certain components and detection heads in an LiDAR object detection DNN to enable the latency and accuracy tradeoffs at runtime. However, such model-level improvements may not work on different model architectures, which are constantly evolving.

In this work, we present versatile anytime algorithm for the LiDAR Object detection (VALO), a novel data-centric approach to enable anytime computing in processing the LiDAR-based object detection DNNs. VALO selectively processes subsets of periodically given input data with the aim of maximizing detection accuracy while respecting the deadline constraint. It implements a deadline-aware scheduler

Manuscript received 10 August 2024; accepted 12 August 2024. This work was supported in part by the NSF under Grant CNS-1815959, Grant CPS-2038923, Grant III-2107200, and Grant CPS-2038658. This article was presented at the International Conference on Embedded Software (EMSOFT) 2024 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Ahmet Soyyigit.)*

Ahmet Soyyigit and Heechul Yun are with the Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS 66045 USA (e-mail: ahmet.soyyigit@ku.edu; heechul.yun@ku.edu).

Shuochao Yao is with the Department of Computer Science, George Mason University, Fairfax, VA 22030 USA (e-mail: shuochao@gmu.edu).

Digital Object Identifier 10.1109/TCAD.2024.3443774

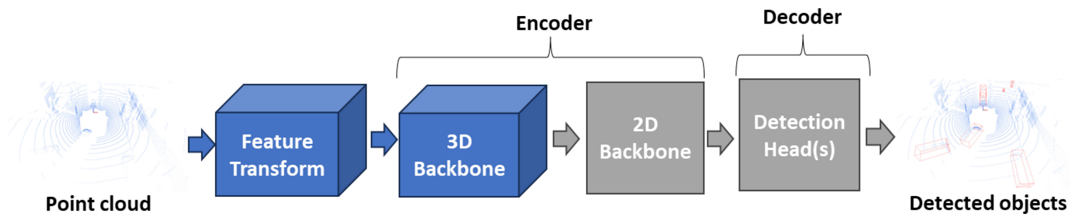


Fig. 1. General LiDAR object detection DNN architecture.

that splits the detection area into regions and schedules them to reduce the computational costs while considering the accuracy impacts. To minimize the potential accuracy loss, VALO employs a lightweight forecasting algorithm to predict the current poses of the previously detected objects based on a simple physics model. The forecasted objects are merged with the DNN detected ones through the nonmaximum suppression to improve the overall accuracy. In addition, VALO implements a novel input reduction technique within its detection heads. This technique reduces the input volume to be processed by a factor of ten for the convolutions responsible for delivering the object attributes. Importantly, it accomplishes this without any loss in accuracy by eliminating the unnecessary computation in the areas where no object prediction exists.

We have implemented VALO on the top of the two state-of-the-art LiDAR object detection DNNs [1], [3] and evaluated them using a large-scale autonomous driving dataset, nuScenes [12]. We utilized the Jetson AGX Xavier [13] as the testing platform, a commercially available off-the-shelf embedded computing platform. The results demonstrate that VALO enables the anytime capability across a wide spectrum of timing constraints, while achieving higher accuracy across all the deadline constraints compared to the baseline LiDar object detection DNNs [1], [3] and a prior anytime approach [11].

In summary, we make the following contributions.

1) We propose a novel data scheduling framework for the LiDAR object detection DNNs that enables latency and accuracy tradeoffs at runtime.
2) We apply our approach to the two state-of-the-art LiDAR object detection DNNs and show its effectiveness and generality on a real platform using a representative autonomous driving dataset.

The remainder of this article is organized as follows. We provide the necessary background in Section II and the present motivation in Section III. We describe our approach in Section IV and present the evaluation results in Section V. After discussing the related work in Section VI, we conclude in Section VII.

II. Background

In this section, we provide the necessary background on the LiDAR object detection DNNs and anytime computing.

A. LiDAR Object Detection DNNs

The primary objective of the LiDAR-based object detection is to identify objects of interest within the detection area by processing the input point clouds. Many LiDAR-based object detection DNNs have been proposed [1], [2], [3], some are optimized for latency, while the others are optimized for accuracy.

Fig. 1 illustrates the general workflow of the LiDAR object detection DNNs. Their encoders are designed to extract features from the transformed input (e.g., voxels) with their backbone(s), typically by employing convolutional neural networks. An encoder can have a 3-D backbone that applies sparse convolutions on the 3-D data, a 2-D backbone similar to those used in vision object detection DNNs or both. When both are used, the sparse output of the 3-D backbone is projected to a bird-eye view (BEV) pseudo image to turn it into a dense tensor so the 2-D backbone can process it with dense convolutions.

After the encoder operation, the produced features are further processed by the decoder, which consists of one or more detection heads to output the 3-D bounding boxes of the identified objects. When multiple detection heads are used, the targeted object classes are separated into groups depending on their size, and each detection head becomes responsible for one group [14]. Within each detection head, a series of convolutions is applied to infer various object attributes, such as location, size, and velocity. Ultimately, nonmaximum suppression or max pooling is used to extract the final results from the predicted candidates.

B. Sparse Convolution

A point cloud P is represented as an array of 3-D point coordinates (x, y, z), each accompanied by attributes, such as LiDAR return intensity i

$$P = \{(x_1, y_1, z_1, i_1), \ldots, (x_n, y_n, z_n, i_n)\}. \quad (1)$$

Unlike 2-D images, the indexes in the array of points do not inherently establish neighborhood relationships, creating a challenge for processing them with commonly used dense convolutional neural networks operating on the dense tensors. To address this issue, the point clouds are transformed into alternative representations, such as a 3-D grid of fixed-size *voxels* created by grouping spatially nearby points [1], [3]. These voxels can be represented as a 3-D dense tensor and processed by 3-D convolutions. However, this approach is avoided due to the significant computational overhead it incurs. Instead, voxels are represented as a sparse tensor and processed by sparse convolutions [15]. A sparse tensor V can be defined in coordinate list (COO) format, where each coordinate has a corresponding array of values. These values represent the features of each coordinate.

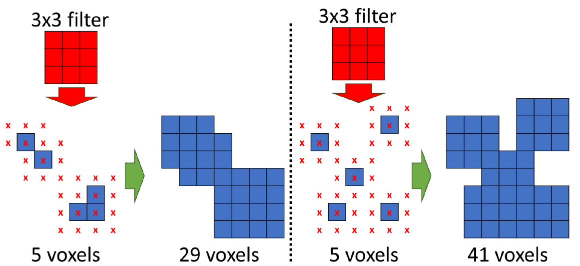


Fig. 2. Two sparse convolution examples applying 3×3 filters. Blue squares indicate voxels. Red markings indicate the coordinates where the filter is applied.

Sparse convolutions can yield the same result as dense convolutions while operating on the sparse tensors. If the input tensor is significantly sparse, as in the LiDAR point clouds, this saves a bulk of computational time compared to the dense convolutions. For this reason, the state-of-the-art LiDAR object detection DNNs commonly employ sparse convolutions. Sparse convolutions apply given filters on all the coordinates where an input coordinate overlaps with any part of the filter.

It is important to note that a sparse convolution operation can generate a differently shaped output tensor, depending on the shape of the input tensor as shown in Fig. 2. As we will discuss in Section IV-C, this introduces input-dependent timing variability in processing the sparse convolutions.

C. Anytime Computing

Anytime algorithms refer to a class of algorithms that can trade deliberation time for the quality of the results [16]. An anytime algorithm is capable of delivering a result whenever it is requested, and the quality of the result improves as the algorithm dedicates more time to finding the solution. For example, a path planning algorithm that progressively enhances its solution by continuously refining the path it has discovered can be considered as an anytime algorithm [16]. In real-time systems, anytime algorithms are highly valuable for meeting dynamically changing deadlines as they can effectively tradeoff between the latency and quality.

Contract algorithms are a special type of anytime algorithms that require a predetermined time budget to be set prior to their activation [17]. They are noninterruptible and deliver results within the time budget, unlike the arbitrarily interruptible anytime algorithms. In deadline-driven real-time systems, such as self-driving cars, the contract algorithms can be used to effectively trade the execution time for accuracy. Providing a framework to transform an LiDAR object detection DNN into a contract algorithm to make it deadline aware is the primary focus of our work.

III. Motivation

To understand the requirements of an effective latency and accuracy trading approach, we profile two representative LiDAR object detection DNNs in detail on the Jetson AGX Xavier.

TABLE I
EXECUTION TIME (MS) STATISTICS OF POINTPILLARS

Stage	Min	Average	99th Perc.	Percentage
Load to GPU	7.99	9.59	10.96	7%
Feature Transform	5.75	6.10	6.42	4%
3D Backbone	5.80	7.07	7.75	5%
Project to BEV	3.13	3.90	4.66	3%
2D Backbone	53.50	53.73	54.15	37%
Detection Heads	56.85	61.27	64.53	44%
End-to-end	136.77	142.07	146.06	100%

TABLE II
EXECUTION TIME (MS) STATISTICS OF CENTERPOINT

Stage	Min	Average	99th Perc.	Percentage
Load to GPU	8.18	9.78	11.28	3%
Feature Transform	3.62	3.83	3.94	1%
3D Backbone	53.64	93.09	134.27	41%
Project to BEV	4.20	4.37	5.60	2%
2D Backbone	70.95	71.24	71.45	21%
Detection Heads	100.91	104.69	106.63	32%
End-to-end	245.83	287.66	329.01	100%

Table I presents the execution time statistics for the PointPillars [2], a well-known LiDAR object detection DNN recognized for its low latency. We observe that approximately 79% of the total processing time is consumed by its 2-D backbone and detection heads. Therefore, a latency-accuracy tradeoff approach targeting these two stages can yield satisfactory results as explored in a recent prior work [11].

However, when the state-of-the-art LiDAR object detection DNNs are considered, an approach that only focuses on the 2-D backbone and detection heads might not be efficient.

Table II shows the execution time breakdown of CenterPoint [1], a recent 3-D LiDAR object detection DNN that achieves higher detection accuracy than the PointPillars [2]. Note that, it spends significantly more time on the 3-D backbone stage, accounting for 41% of the total execution time.

Although adopting sparse convolutions partially alleviates the computational burden of the 3-D backbone [15], [18], it still demands significant computational resources. Thus, the 3-D backbone becomes another computational bottleneck, which must be addressed when trading the accuracy for lower latency.

One simple approach for achieving the latency-accuracy tradeoff is training multiple models with varying input granularity (i.e., resolutions) and dynamically switching between them. However, this approach can be cumbersome during runtime due to the overhead involved in the model switching (in terms of the memory overhead and switching latency). It also necessitates training and fine tuning a large number of models to achieve finely tuned tradeoffs.

Instead, we focus on developing a single model that can deliver the highest possible accuracy when there is flexibility with the deadline, while intelligently adjusting input data when the deadline becomes more stringent, as will be discussed in the next section.

IV. VALO

In this section, we introduce VALO, a scheduling framework that transforms aN LiDAR object detection DNN into a

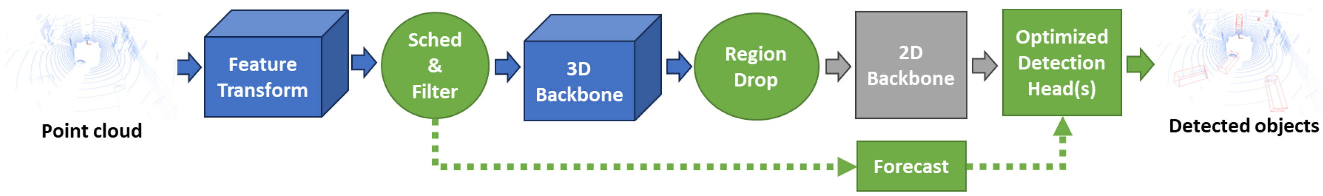


Fig. 3. Overview of VALO.

noninterruptable anytime (contract) algorithm. VALO allows detection results to be produced in time for a gamut of deadline requirements, with a controlled tradeoff in accuracy.

A. *Overview*

The fundamental concept underpinning VALO's design is the *scheduling of data* to facilitate the tradeoffs between the time and accuracy rather than scheduling the architectural components of the targeted DNN. This design choice makes VALO versatile, as it is not constrained by the architectural specifics of the LiDAR object detection DNNs. Fig. 3 illustrates VALO's three main components: 1) scheduling; 2) forecasting; and 3) detection head optimization, highlighted in green, and their positions within the DNN pipeline. The region drop component is considered a part of scheduling.

First, VALO's scheduler comes into play after the DNN has completed the feature transformation stage. This allows it to make scheduling decisions at the voxel level instead of the raw point clouds, enabling more accurate predictions of the timing for the 3-D backbone stage.

During the scheduling phase, VALO decides which regions of the input data will be processed to maximize detection accuracy within the deadline constraint. Once a decision is made, the data outside the selected regions is filtered out, and the remaining data is forwarded to the subsequent stage (Section IV-B).

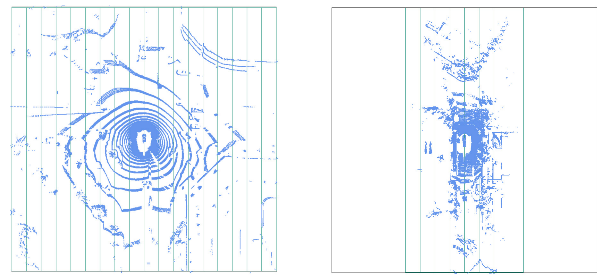
For effective region scheduling, VALO predicts execution times of subsequent network stages of each possible region selection (Section IV-C). VALO also employs a mechanism to recover from the execution time mispredictions (Section IV-D).

Next, while filtering part of the input can reduce latency, it can also negatively impact accuracy. To mitigate potential accuracy loss, VALO employs a forecasting mechanism that updates the positions of the previously detected objects to the current time of execution. This operation is performed mostly in parallel while the DNN executes. After the, detection heads generate object proposals, these proposals are combined with the list of forecasted objects. The combined list is then subjected to nonmaximum suppression, which yields the final detection results (Section IV-E).

Finally, to further improve the efficiency, we introduce a novel optimization technique for the efficient detection head processing. This optimization technique eliminates the significant amount of redundant computation in detection heads without compromising the detection accuracy (Section IV-F).

B. *Region Scheduling*

The scheduler decides which subset of input data (voxels) should be processed to meet a given deadline while maximizing the accuracy. Intuitively, the less data it selects, the less time it takes for the DNN to process it, albeit at the expense of reduced accuracy. To make the scheduling problem tractable, we partition the fixed-size detection area into equally sized chunks along the X (width) axis, which we refer to as *regions*.



(a) (b)

Fig. 4. Two examples of how the region scheduler partitions the detection area into regions. (a) Partitioning example 1. (b) Partitioning example 2.

Fig. 4 illustrates two examples of partitioning a 108×108 m^2 detection area into 18 vertical regions. In Fig. 4(a), the input point cloud is spread to all 18 regions. In contrast, Fig. 4(b) shows that only a portion of the regions, 8 out of 18, contain points due to the structure of the environment scanned by LiDAR. In scenarios with empty regions, the scheduler skips all the empty regions located before the first nonempty region and after the final nonempty region. As a result, partitioning the input in the X axis for some inputs allows for latency reduction without sacrificing accuracy in later stages.

To determine which regions to process, we employ a greedy policy that sequentially selects the maximum number of input regions while adhering to the deadline constraint. Consequently, all regions are treated with equal priority. Fig. 5 provides an illustrative example of the proposed region scheduling algorithm, which selects regions for processing over three consecutive inputs. For each input, the scheduler decides the regions to be scheduled for processing, starting from the next to the final of the previously scheduled regions, which can meet the given deadline.

Algorithm 1 outlines our proposed scheduling algorithm. Initially, the scheduler counts the number of voxels in each region and returns the list of schedulable regions (R_S), and their voxel counts (C_S) (line 8).

The scheduler then reorders the obtained list so the selections start from the first nonempty region coming after r_{last} (lines 9). Subsequently, candidate region selections are iterated from largest to smallest until one that meets the deadline is identified (lines 10–18).

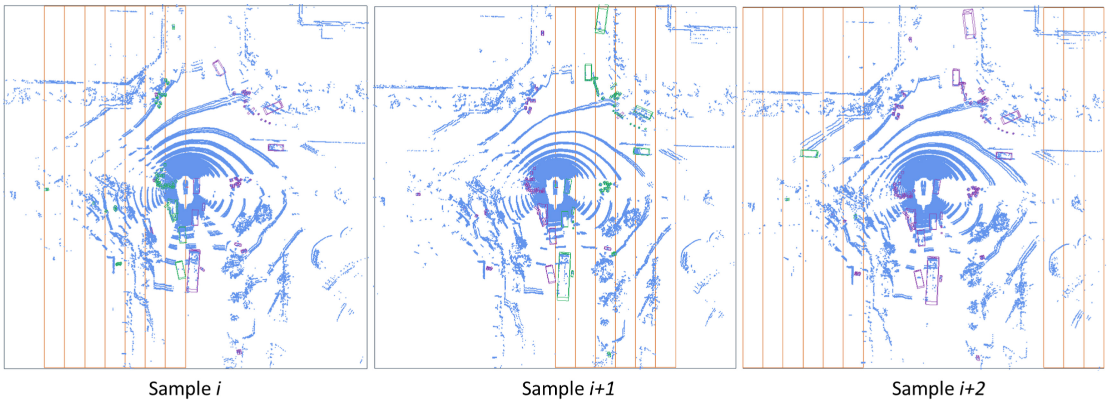


Fig. 5. Example of region scheduling on three consecutive samples over time. The regions outlined in orange represent the selections made by the scheduler for processing. The green and purple bounding boxes indicate the objects detected as a result of processing the selected regions and the forecasted objects, respectively. Best viewed in color.

Once scheduling is completed, input voxels falling outside the selected regions (R_{sel}) are filtered, and the remaining voxels are forwarded to the 3-D backbone as input. If the subsequent stage employs dense convolutions, the sparse output of the 3-D backbone is then converted to a dense tensor where the regions are placed following the order in R_{sel}.

Our scheduling method brings three advantages. First, selecting the adjacent regions maintains spatial continuity and processes the input with minimal fragmentation, thereby avoiding accuracy degradation that can happen through slicing and batching nonadjacent regions. Second, it ensures a consistent level of "freshness" of object detection results over all the regions, which is needed for effective forecasting operations (Section IV-E). Third, it incurs minimal scheduling overhead.

Algorithm 1: Scheduling Algorithm

```
1  Input:
2  Input voxels (V),
3  Number of input regions (N_R),
4  Last scheduled region (r_last),
5  Relative deadline (D),
6  Output: Selected regions to be processed
7  function schedule(V, N_R, r_last, D)
8      R_S, C_S ← count_voxels(V, N_R)
9      R_S, C_S ← reorder(R_S, C_S, r_last)
10     i ← length_of(R_S)
11     while i ≥ 1 do
12         R_sel, C_sel ← R_S[:i], C_S[:i]
13         E ← calc_wcet(R_sel, C_sel)
14         rem_time ← D − get_elapsed_time()
15         if E < rem_time then
16             i ← 0
17         else
18             i ← i − 1
19 return R_sel
```

C. Execution Time Prediction

For effective region scheduling, the key *challenge* is to determine whether a candidate list of regions can be processed within a given deadline constraint (line 13 in Algorithm 1). The predicted execution time E of a candidate list of regions can be calculated as

$$E = E_S + E_D + E_R \tag{2}$$

where E_S is the time to process sparse data (i.e., 3-D backbone), E_D is the time to process dense data (i.e., 2-D backbone and convolutions in detection heads), and E_R is the time to process the final stage of object detection task, such as nonmaximum suppression.

For E_D, since the number of candidate regions ($|R_{\text{sel}}|$) determines the size of the dense input tensor that will be passed to the 2-D backbone, it can be defined as an one-to-one function, where each possible $|R_{\text{sel}}|$ is mapped to an execution time determined through the offline profiling. This mapping is feasible because the execution time of dense convolutions remains largely fixed as a function of input size, and there is a small finite number of possible regions.

On the other hand, E_S, the execution time of the sparse 3-D backbone, is difficult to predict as it depends on the number of input voxels in a highly nonlinear manner as shown in Fig. 6.

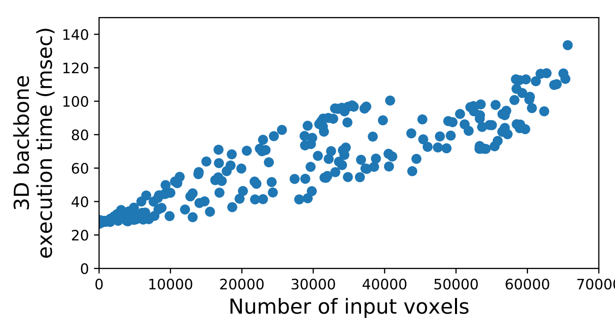


Fig. 6. Profiled execution time CenterPoint's 3-D Backbone.

This nonlinearity mainly stems from the fact that a sparse convolution layer can generate a different number of output voxels for the same number of input voxels depending on their relative positions as illustrated in Fig. 2. Consequently,

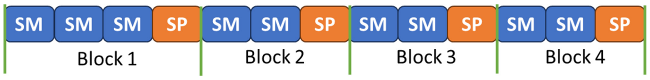


Fig. 7. CenterPoint's 3-D backbone broken into blocks. SM: submanifold sparse convolution. SP: sparse convolution.

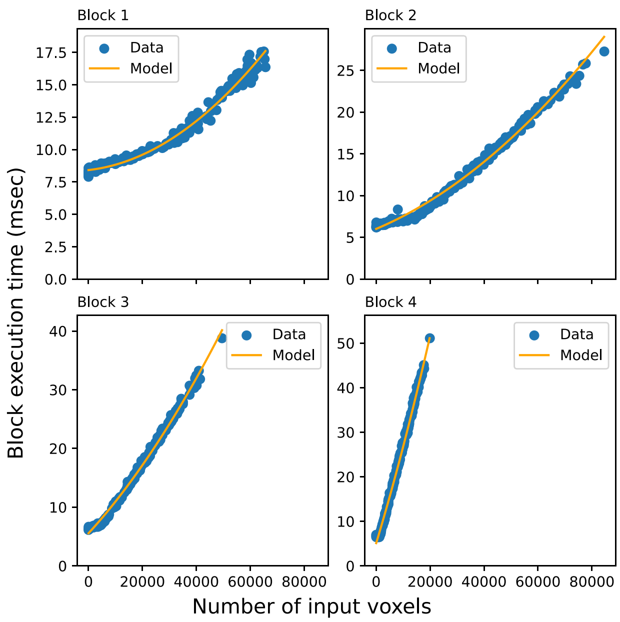


Fig. 8. Profiled execution time of the blocks of CenterPoint's 3-D backbone and the quadratic models regressed from their execution times data.

the computational demand of processing a subsequent layer, which takes the output of the previous layer as input will vary accordingly. To make the time prediction tractable, we break the 3-D backbone into blocks at points where the count of forwarded voxels changes as illustrated in Fig. 7.

We then focus on separately predicting the execution time of each block. Note that, unlike a sparse convolution layer, batch normalization, activation functions, and submanifold sparse convolution [19], all of which heavily used in 3-D backbones, do maintain the same input and output shapes (thus the voxel counts), and thus can be safely grouped within a block. Denoting V_i as the input voxels of a layer L_i, we define a block B as

$$B = \{L_k, \ldots, L_l \mid \forall i, \quad k \le i \le l, \quad |V_k| = |V_i|\} \tag{3}$$

where V_k is the input voxels of the first layer L_k. The input of a block B denoted as V_B is the same as V_k.

Fig. 8 shows the execution time profiles of all the four blocks of the CenterPoint's 3-D backbone. As can be seen in the figure, each block's execution time, as a function of the number of input voxels of the block, is more predictable using a simple quadratic prediction model

$$E_{B_i}(|V_{B_i}|) = \alpha |V_{B_i}|^2 + \beta |V_{B_i}| + \gamma \tag{4}$$

where the coefficients α, β, and γ are determined by regression against the profiling data collected offline. Then, the execution time of the 3-D backbone can be predicted as follows:

$$E_S = \sum_{i=1}^{n} E_{B_i}(|V_{B_i}|). \tag{5}$$

However, a major challenge is that, except for the first block, the number of input voxels of the remainder of the blocks, C_{rest}, are not known until the execution of the preceding blocks is completed

$$C_{\text{rest}} = \left\{|V_{B_2}|, \ldots, |V_{B_n}|\right\}. \tag{6}$$

To predict C_{rest} for any given list of candidate input regions, we use a history-based approach, leveraging the fact that there is a strong similarity between the consecutive LiDAR scans, as the movements of objects between the scans are limited. Specifically, for the block B_2 to B_n, we keep track of each block's most recent input voxel counts of all the input regions, which are updated whenever they are selected by the region scheduler and processed. Assuming voxel counts would be similar over time, we then aggregate the latest voxel counts of the current candidate regions to obtain C_{rest}.

Finally, for E_R, the execution time to perform nonmaximum suppression and other operations can vary depending on the number of object proposals in the detection pipeline. However, because it is relatively small compared to the remainder of the pipeline, namely E_D and E_S, we simply use the 99th percentile of the measured execution time through offline profiling, which provides a safe upper bound without significantly affecting the time prediction accuracy.

D. Region Drop

The aforementioned execution time prediction method for the 3-D backbone can inevitably introduce some inaccuracy. For LiDAR object detection models with 2-D backbones, such as CenterPoint [1], after the execution of the 3-D backbone, we additionally check if it will be possible to meet the deadline (see Fig. 3), considering the predicted execution time of the remainder of the pipeline. If deemed not possible, we further reduce the number of input regions so that the deadline can be met. Note, however, that some recently proposed LiDAR object detection models, such as VoxelNext [3] do not employ a 2-D backbone as they are fully sparse. For such networks, the region dropping does not apply.

E. Forecasting

Forecasting estimates the present *pose* of the objects identified in the past invocations of the object detector. Because our region scheduling method (Section IV-B) can skip part of the input LiDAR scan due to the deadline constraints, forecasting plays a critical role in mitigating the potential accuracy loss.

We define a pose P of an object at time t as

$$P_t = \{T, S, \alpha, v, c, l\} \tag{7}$$

where T is the 3-D coordinate of the object expressed in the LiDAR coordinate frame, S is the bounding box, α is the heading angle, v is the velocity vector, c is the confidence score, and l is the label (e.g., car or pedestrian). In this work,

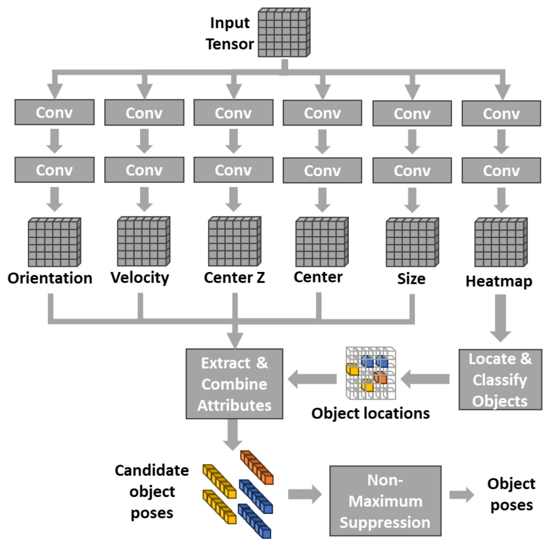


Fig. 9. General detection head architecture.

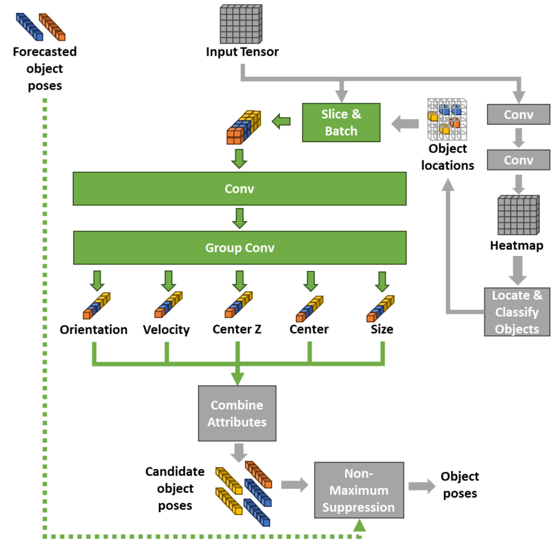


Fig. 10. Optimized detection head architecture.

we focus on estimating T and α and assume the others to stay consistent over time.

The first part of forecasting involves maintaining a queue of previously detected object poses. For all the processed input regions of an input frame, VALO removes the old objects corresponding to the processed regions from the queue and appends the freshly detected objects in these regions to the queue. Thus, the queue maintains the latest detected objects of all the regions.

The second part of forecasting involves performing the mathematical calculations to estimate $P_{t_{\text{cur}}}$ for all the objects in the pose queue. For each pose of an object in the pose queue, we first rotate and translate the object pose to be expressed in the global coordinate frame using the ego-vehicle pose. We then add the distance traveled by the object ($v \times (t_{\text{cur}} - t_{\text{det}})$) to the translation component (T) of the pose. Finally, we translate and rotate the pose to be expressed in the current LiDAR coordinate frame.

At the runtime, we update the queue on the CPU and perform the actual pose updates on the GPU. We have developed a custom GPU kernel to update the poses of all the objects in parallel. The forecasting GPU kernel is executed in a separate CUDA stream to maximize the parallelism.

F. Detection Head Optimization

LiDAR object detection DNNs include detection heads that are designed to extract specific attributes of objects, such as position, size, and orientation. Surprisingly, we discovered that a significant amount of redundant computations occur in processing the detection heads of the state-of-the-art LiDAR object detection DNNs [20].

Fig. 9 illustrates the general architecture of a detection head, which performs a series of convolutions to infer attributes of the objects. The width and height dimensions of the output tensors from these convolutions correspond to the width and height of the detection area in the BEV. Among the inferred attributes, the *heatmap* plays the most important role, as it holds the confidence scores of the objects used for classifying and locating them. In a heatmap tensor, any score value above a predefined score threshold indicates an object proposal. The list of object proposals, R, extracted from the heatmap can be expressed as

$$R = \{(c_1, x_1, y_1), \ldots, (c_n, x_n, y_n)\} \tag{8}$$

where c is the confidence score and x and y are a position in the detection area. Once R is generated, remaining object attributes (e.g., orientation, velocity, size, etc.) are obtained from their corresponding output tensors at the x and y positions in R, and combined into object poses (7).

The problem with this approach is that it performs convolutions on all the parts of the input while only the output locations that correspond to the object proposals (R) are utilized. As a result, the convolutions inferring object attributes except the heatmap involve a significant amount of redundant computation.

To improve efficiency, we propose to optimize the detection head processing as follows.

1) The heatmap is computed in the same manner as in the baseline approach.
2) The detected object list R from the heatmap is utilized to selectively gather and batch small patches from the input tensor.
3) Convolutions are applied to this batch of patches to derive the object attributes.

Fig. 10 provides a visual representation of the proposed approach. Note that, the proposed optimization ensures that convolutions are applied only to the data that is needed for producing the desired output corresponding to the locations in R. This approach significantly reduces the number of multiply accumulate operations (MACs) without any loss of detection accuracy.

However, due to the reduction in the input size, there is a potential issue of GPU underutilization if we execute the attribute-inferring convolutions one by one as in the baseline. To maximize GPU utilization, we concatenate them into a single convolution operation followed by a group convolution. This improves GPU utilization and reduces the GPU kernel invocation overhead.

Note that, some recent LiDAR object detection networks, such as VoxelNext [3] employ sparse convolutions in detection heads instead of dense convolutions. For such a model, we replace the slice and batch part of detection head optimization with filtering all the sparse tensor coordinates that do not contribute to the output, and do not group the convolutions as they are sparse. In this way, we significantly reduce computational overhead without losing detection accuracy and allow utilizing of the model trained for the baseline.

V. Evaluation

For evaluation, we implemented VALO as an extension to OpenPCDet [20], an open-source framework for LiDAR 3-D object detection DNNs, which supports the state-of-the-art methods. For this study, we mainly target CenterPoint [1] as a baseline and apply VALO to demonstrate its effectiveness. In addition, we also apply VALO on a more recently proposed VoxelNext [3], a fully sparse DNN, to demonstrate the versatility of our approach.

As for the dataset, we utilize nuScenes [12], a large-scale autonomous driving dataset, and use the nuScenes detection score (NDS) [12] as the detection accuracy metric since it was reported to correlate with the driving performance better than the classic average precision (AP) metric [21]. In the remainder of the evaluation, unless noted otherwise, we normalize the NDS score with respect to the maximum NDS score we observed among the all compared methods. We utilize 30 distinct scenes from the nuScenes evaluation dataset, with each scene containing annotated LiDAR scans spanning 20 s, sampled at intervals of 350 ms. The sample period is chosen to match the worst-case execution time of the slowest baseline method on our evaluation platform.

To capture the timeliness aspect of the detection performance, we evaluated the methods under a range of deadline constraints from 350 to 90 ms. The deadline range is chosen to be between the best-case execution time of the fastest baseline method and the worst-case execution time of the slowest baseline model. During each test, we kept a buffer holding the latest detection results and updated this buffer every time the method being tested met the deadline. In case of a deadline miss, we considered the buffered detection results as the output and ignored the produced ones by assuming the job was aborted.

As for the hardware platform, we used an NVIDIA Jetson AGX Xavier [13], equipped with 16 GiBs of RAM for the runtime performance evaluation. We maximized all the hardware clocks and allocated the GPU resources only for the method being tested. For software, we used Jetson JetPack 5.1 and Ubuntu 20.04. Training of the models was done on a separate desktop machine with an NVIDIA RTX 4090 GPU.

We present the evaluation results in the following three subsections. First, we compare VALO with a set of baselines to evaluate its performance. Second, we perform an ablation study to demonstrate the benefits of VALO's components. Finally, we shift our focus to the intrinsic details of VALO and analyze the execution time behavior of its components.

A. Comparison With the Baselines

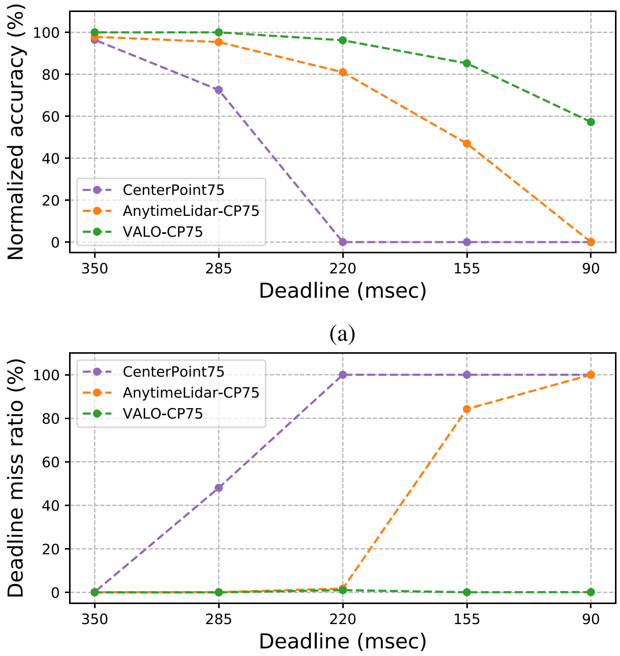
Below is the list of methods we compared in this section.

1) *CenterPoint [1]:* This is a representative state-of-the-art LiDAR object detection network architecture that employs a voxel encoder as its 3-D backbone, followed by a region-proposal-based 2-D backbone and six detection heads, each of which focuses on a subset of the object classes [14]. Before being forwarded to the 3-D backbone, the input point cloud is transformed into fixed-sized voxels. The size of a voxel is a design parameter of the network, which should stay consistent during training and testing. In this work, we consider three voxel configurations $75 \times 75 \times 200$ mm^3, $100 \times 100 \times 200$ mm^3, and $200 \times 200 \times 200$ mm^3, which are called CenterPoint75, 100, and 200, respectively. Employing bigger voxels reduces the computing cost at the expense of accuracy.
2) *VoxelNext [3]:* A recently proposed LiDAR object detection network, featuring a voxel encoder as its 3-D backbone deeper than the CenterPoint's followed by six detection heads. Unlike CenterPoint, all the convolutions in its detection heads operate on the sparse tensors. Like CenterPoint, VoxelNext also can be configured to have a different voxel size. We focus only on the setting that employs voxels of size $75 \times 75 \times 200$ mm^3 (i.e., VoxelNext75).
3) *AnytimeLidar [11]:* To the best of our knowledge, this is the only work that can provide runtime latency and accuracy tradeoff (i.e., anytime computing) for the LiDAR object detection DNNs in the literature. It achieves the anytime capability by utilizing early exits in processing the 2-D backbone and skipping a subset of the detection heads dynamically. While AnytimeLidar is originally based on the PointPillars [2], we ported it to the CenterPoint75 baseline to make a fair comparison, which we call AnytimeLidar-CP75. Note that, AnytimeLidar cannot be applied to the VoxelNext since it lacks a 2-D backbone.
4) *VALO:* The proposed method in this work. VALO can be applied to the CenterPoint and VoxelNext baselines. We call VALO-CP75 and VALO-VN75 when it is applied to the CenterPoint75 and VoxelNext75 baselines, respectively.

1) VALO Versus AnytimeLidar: In this experiment, we compare the performance of VALO and AnytimeLidar with the CenterPoint75 baseline from which they are applied.

Fig. 11 shows the results. Fig. 11(a) compare how detection accuracy changes in relation to the varying deadline constraints. Fig. 11(b), on the other hand, compare the corresponding deadline miss rates of the tested methods under the deadline constraints.

Note first that, under the 350 ms deadline constraint, all the methods can meet the deadline without a need for the tradeoffs and demonstrate their maximum accuracy. When the



(a)

(b)

Fig. 11. VALO versus AnytimeLidar on CenterPoint. (a) Detection accuracies. (b) Deadline miss rates.

deadline tightens, however, the CenterPoint baseline immediately begins to miss deadlines as it cannot adjust its computing demand according to the given deadline, resulting in a significant drop in accuracy. AnytimeLidar and VALO, on the other hand, can trade accuracy for lower latency (i.e., anytime capable), and thus achieve improved performance as they can meet the deadlines better. However, when the deadline is 155 ms, AnytimeLidar starts to miss deadlines due to its limited anytime computing capability. But VALO respects the deadline constraints down to 90 ms and achieves higher accuracy.

AnytimeLidar falls short of matching the effectiveness of VALO primarily due to dismissing the contribution of the 3-D backbone on the total latency. Moreover, AnytimeLidar's effectiveness will be further reduced if a single detection head architecture, instead of the multihead detection architecture in this work, is used because its ability to make a tradeoff is in large part enabled by skipping a subset of the detection heads, which is possible only in the multihead architecture.

In contrast, VALO can make fine-grained execution time and accuracy tradeoffs, primarily due to its ability to schedule a portion of the data to process, independent of the neural network architectural specifics, such as 3-D/2-D backbone or the number of detection heads. This distinct focus on the data makes VALO a more versatile framework that can be applied in any LiDAR object detection DNN.

2) VALO Versus Other Nonanytime Baselines: Fig. 12 shows the detection performance of VALO-CP75 and three other CenterPoint baselines. All the baselines have distinct execution time demands and accuracy they can deliver. For example, when the deadline is 350 ms, CenterPoint75 achieves the best accuracy among the three baselines. But when the deadline is 220 ms, CenterPoint75's accuracy falls down to zero because it no longer is able to meet the deadline. On the other hand, CenterPoint200's accuracy does not change all the way down to the deadline of 155 ms as it can still meet the deadline albeit at a somewhat lower accuracy. Note, however, that these baseline models are fixed and cannot make accuracy versus latency tradeoffs on the fly at runtime. VALO, on the other hand, can adapt itself to a wide range of deadline constraints from 90 to 350 ms on the fly while providing the best possible accuracy for a given deadline constraint.

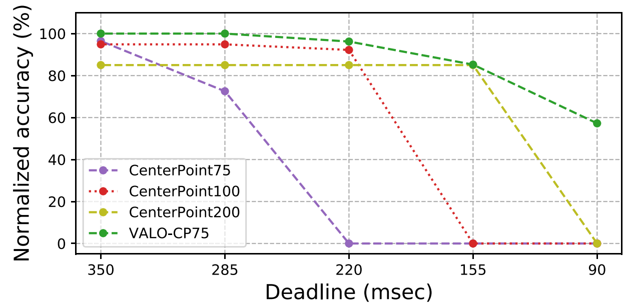


Fig. 12. VALO versus CenterPoint variants.

As an alternative way to adapt to the varying deadline constraints on the fly, one can consider using multiple DNN models of differing latency-accuracy tradeoffs (like CenterPoint75, 100, and 200 in this experiment) and switch between them depending on a given deadline constraint at runtime as done in [4]. However, the problems of such an approach are that it needs to train, fine-tune, and manage all these models separately. Furthermore, these models need to be loaded into the precious (GPU) memory all the time for the real-time operations, even when only one of them is actually used at a time. In contrast, VALO can make such tradeoffs at runtime from a single model without requiring any additional memory overhead.

3) VALO on VoxelNext: To demonstrate VALO's versatility, we applied it to the VoxelNext [3], which has a significantly different architecture than the CenterPoint. Unlike CenterPoint, VoxelNext does not use a 2-D backbone and instead relies solely on the 3-D sparse convolution layers.

Fig. 13 shows the result. As in the CenterPoint case, VALO-VN75 performs better than the baselines in all the deadline constraints. The region scheduling (Section IV-B) allows VALO-VN75 to dynamically adjust the time spent on the 3-D backbone and the detection heads effectively, effectively making it anytime capable.

4) Effectiveness of Time Prediction: The effectiveness of VALO's region scheduling critically depends on the accuracy of its time prediction (Section IV-C). To evaluate the effectiveness of the proposed history-based time prediction method, we compare its accuracy with a simple quadratic prediction model that directly predicts the execution time of the entire 3-D backbone from the number of input voxels (as opposed to predicting per block-based prediction in our proposed history-based time prediction approach). We denote this baseline method as *quadratic* whereas our history-based approach as *history*.

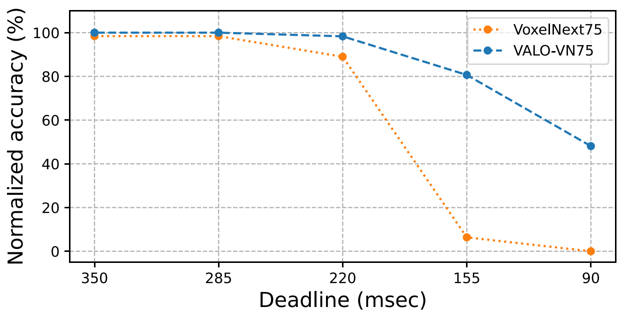


Fig. 13. VALO on VoxelNext.

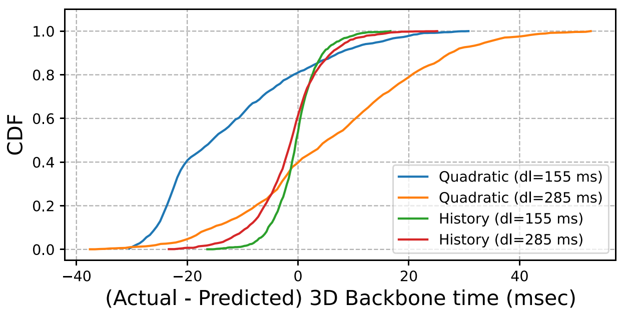


Fig. 14. Cumulative distribution function of time prediction error for history-based and baseline methods.

Fig. 14 compares the accuracy of both time prediction methods in predicting 3-D backbone execution time against the evaluation dataset. As can be seen in the figure, our history-based prediction method significantly outperforms the baseline quadratic method, which helps reduce deadline violations and improve detection accuracy.

B. Ablation Study

In this experiment, we investigate the contribution of region scheduling and forecasting by comparing VALO with its two variants explained below. We also include the CenterPoint75 baseline for comparison.

1) *VALO-NSNF-CP75:* This variant of VALO operates without scheduling (Section IV-B) and forecasting (Section IV-E), hence denoted as "no scheduling no forecasting" (NSNF). However, it does perform detection head optimization (Section IV-F).
2) *VALO-NF-CP75:* This variant of VALO performs region scheduling (Section IV-B) and detection head optimization (Section IV-F), but not forecasting (Section IV-E).

Fig. 15 presents the experimental results where we observe improved performance as additional VALO components are introduced to the baseline CenterPoint75. First, VALO-NSNF-CP75 achieves a higher accuracy over the baseline CenterPoint75 when the deadline is tighter than 350 ms. For instance, at the 285 ms deadline, VALO-NSNF-CP75 matches the accuracy of CenterPoint75 at 350 ms. This underscores the effectiveness of the detection head optimization in reducing the execution time without compromising accuracy. Next, VALO-NF further improves accuracy across a wider range of deadline constraints by enabling region scheduling because it can make

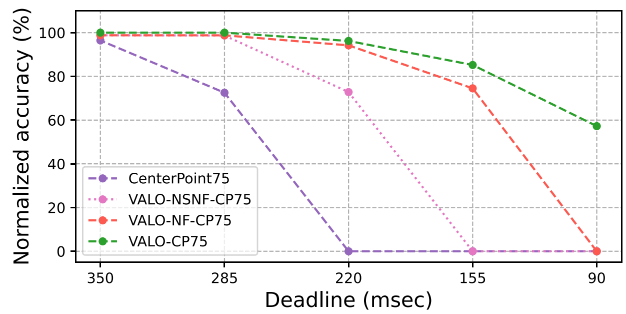


Fig. 15. Detection accuracy achieved by the variants of VALO.

execution time and accuracy tradeoffs, preventing deadline misses and boosting accuracy over VALO-NSNF. Finally, VALO achieves the highest accuracy across all the deadline constraints by additionally utilizing forecasting, which is particularly effective on the tight deadlines. This is because forecasting plays a more crucial role when the number of scheduled regions reduces as the deadline tightens.

C. Component-Level Timing Analysis

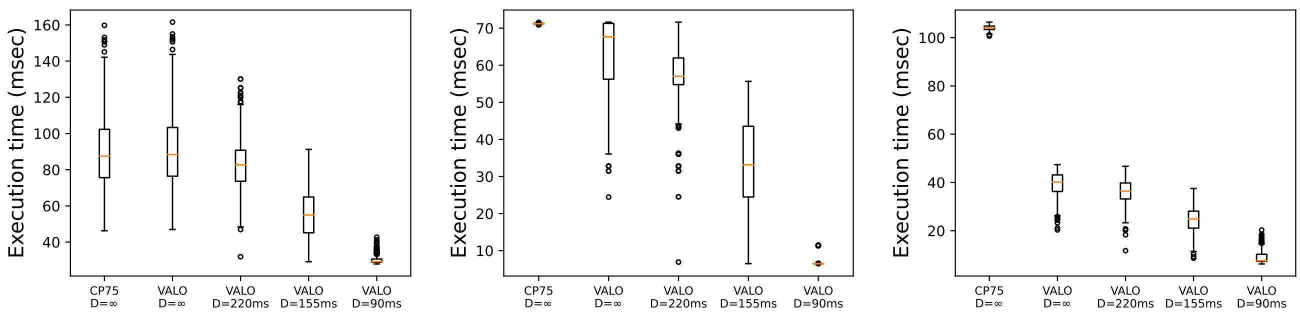
In this experiment, we delve into the execution timing characteristics of the components of VALO when it is applied to the CenterPoint75.

Fig. 16 shows the execution timing of the 3-D backbone, 2-D backbone, and detection heads. For each component, we consider five different cases. The first two involve using CenterPoint75 and VALO-CP75, where there is no deadline. The remainder are the results of VALO-CP75 executed with 220, 115, and 90-ms deadline constraints, respectively.

1) 3-D Backbone: Fig. 16(a) shows the execution time profile of the 3-D backbone portion of the network. Note first that the CenterPoint75 baseline shows a high degree of variations, influenced by the varying count and positioning of the input voxels. When there is no deadline, the time spent on the 3-D backbone of VALO-CP75 is about the same as CenterPoint75 as expected. As the deadline gets tighter, however, VALO's execution time of the 3-D backbone is progressively reduced because its region scheduler dynamically selects a subset of input regions that can be executed within the given time budget.

2) 2-D Backbone: Fig. 16(b) shows the execution time profile of processing the 2-D backbone, where the convolutions on the dense tensors take place. Unlike the 3-D backbone processing, even when there is no deadline, we can observe a notable decrease in the execution time in VALO compared to the CenterPoint75 baseline. This is because our data partitioning scheme (Section IV-B), which exploits the sparsity of the LiDAR data, can skip empty input regions in the 2-D backbone, thus reducing latency. As the deadline get tighter, we also observe a further reduction in the execution time of the 2-D backbone as a result of reduced input data selected by the scheduler.

3) Detection Heads: Fig. 16(c) shows the execution time profile of processing the detection head. Note first that, we observe more than 50% reduction in detection head processing latency on VALO-CP75 compared to the CenterPoint75



(a) (b) (c)

Fig. 16. Component-level execution time profile of the baseline and VALO on Centerpoint75 under different deadline constraints. (a) 3-D backbone (Voxel encoder). (b) 2-D backbone (RPN). (c) Detection head (CenterHead).

baseline even when there is no deadline constraint. This is due to the proposed detection head optimization described in Section IV-F, which significantly reduce the amount of data to be processed by eliminating the redundant data. In addition, as the deadline get tighter, we again observe progressive reduction in the execution time in VALO due to further reduction in the input data to the detection head thanks to its scheduler.

4) Overhead: We measured 3 ms of scheduling overhead in the worst case, including the input filtering time. There is also 3 ms overhead due to the voxel counting operations as a part of history-based time prediction. We did not observe any overhead incurred by the forecasting operation when the end-to-end latency is considered, as it is efficiently executed in parallel with the backbones. Note that, the total overhead of VALO on the CenterPoint75 is only about 6 ms, which is less than 2% of the average execution time of the CenterPoint75.

VI. Related Work

Timely execution of autonomous driving software is essential to ensure safe and efficient navigation. Traditionally, the timing requirements (i.e., deadlines) of the autonomous driving tasks are often fixed at the design time [22], [23], which is not adaptable to the highly varying execution time demands [24]. Recently, Gog et al. [4] have highlighted the potential benefits of adopting a flexible approach, which can dynamically change deadlines in the autonomous driving software based on the specific driving situation, such as the speed of the vehicle or sudden pedestrian appearance to improve the performance and safety of the vehicle.

LiDAR object detection is a critical component in many autonomous driving systems [25]. With the release of large-scale autonomous driving datasets [12], [26], researchers have developed deep learning-based object detection models that achieve the state-of-the-art performance. Besides aiming to achieve high accuracy, the recent work has also considered reducing latency as an objective [1], [2], [3], [27], [28], [29], [30] for the real-time operation. These works can achieve remarkable accuracy in real time when executed on high-end GPUs and accelerators. However, their deployment on the edge computing platforms, such as Jetson AGX Xavier [13] still poses a challenge due to their significant computational overhead and latency. More importantly, they lack the capability to dynamically adapt their execution time in a deadline-aware manner, which is needed for the real-time cyber–physical systems.

Recent studies have explored the concept of "anytime perception" for the neural networks, which enables them to execute within defined deadlines while making trade-offs between execution time and accuracy. For example, Kim et al. [6] achieved this by iteratively adding layers to an image classification network and retraining it to incorporate "early exits." Lee and Nirjon [31] focused on the neuron level, prioritizing critical neurons for accuracy while deactivating the others to save time. Bateni and Liu [7] used perlayer approximation instead of early exits and presented a scheduling solution for the multiple DNN tasks. Yao et al. [8] also dealt with the scheduling of multiple DNN tasks, utilizing imprecise computation alongside early exits. While these works primarily targeted image classification tasks, object detection tasks present unique challenges.

Heo et al. [32] introduced a multipath DNN architecture designed for anytime perception in vision-based object detection. Another work by the same Heo et al. [33] designed an adaptive image scaling method that respects the deadline constraints for the multicamera object detection task. Gog et al. [34] proposed to switch between the DNNs to make latency and accuracy tradeoffs dynamically at runtime. Hu et al. [35] suggested reducing the resolution of less critical parts of the scene to lower computational costs. Lie et al. [9], [36] divided individual image frames into smaller subregions with varying levels of criticality, using the LiDAR data to batch-process essential subregions to meet deadlines. However, these prior efforts mainly focus on 2-D vision and do not account for the unique characteristics of the 3-D point cloud processing.

Recently, Soyyigit et al. [11] proposed a set of techniques that enable anytime capability for the LiDAR object detection DNNs. They focused on the object detection models where the bulk of the computation is performed on the 2-D backbone and detection heads, such as PointPillar [2] and Pillarnet [27]. However, the effectiveness of their approach diminishes on the recent state-of-the-art object detection models where the bulk of time is spent on the 3-D backbone [1], [3]. Fundamentally, such effort that focuses on the model-level improvements may fail to work when the architecture of the model changes.

In contrast, our work focuses on the data-level scheduling, independent of the architectural details of the backbones and detection heads, and thus can be seamlessly applied to any state-of-the-art LiDAR object detection DNNs.

VII. Conclusion

In this work, we presented VALO, a versatile anytime computing framework for the LiDAR object detection DNNs. VALO's superior performance compared to the prior state-of-the-art comes from three major contributions: 1) partitioning the input data into regions and efficiently scheduling them with the goal of maximizing accuracy while respecting the deadlines; 2) lightweight forecasting of the previously detected objects to mitigate the potential accuracy loss due to partially processing the input; and 3) and intelligently reducing redundant computations in processing the detection heads of the object detection neural network with no loss of accuracy. Evaluation results have shown that our approach can adapt to a wide-range of deadline constraints in processing the LiDAR object detection DNNs, and enables a fine grained and effective execution time and accuracy tradeoff.

References

[1] T. Yin, X. Zhou, and P. Krähenbühl, "Center-based 3D object detection and tracking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 1–10.

[2] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 1–9.

[3] Y. Chen, J. Liu, X. Zhang, X. Qi, and J. Jia, "VoxelNeXt: Fully sparse VoxelNet for 3D object detection and tracking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2023, pp. 1–10.

[4] I. Gog, S. Kalra, P. Schafhalter, J. E. Gonzalez, and I. Stoica, "D3: A dynamic deadline-driven approach for building autonomous vehicles," in *Proc. Eur. Conf. Comput. Syst. (EuroSys)*, 2022, pp. 453–471.

[5] Z. Li, T. Ren, X. He, and C. Liu, "RED: A systematic real-time scheduling approach for robotic environmental dynamics," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2023, pp. 210–223.

[6] J.-E. Kim, R. Bradford, and Z. Shao, "AnytimeNet: Controlling time-quality tradeoffs in deep neural network architectures," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2020, pp. 945–950.

[7] S. Bateni and C. Liu, "ApNet: Approximation-aware real-time neural network," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2018, pp. 67–79.

[8] S. Yao et al., "Scheduling real-time deep learning services as imprecise computations," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, 2020, pp. 1–10.

[9] S. Liu et al., "Real-time task scheduling for machine perception in intelligent cyber-physical systems," *IEEE Trans. Comput.*, vol. 71, no. 8, pp. 1770–1783, Aug. 2022.

[10] J.-E. Kim, R. Bradford, M.-K. Yoon, and Z. Shao, "ABC: Abstract prediction before concreteness," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2020, pp. 1103–1108.

[11] A. Soyyigit, S. Yao, and H. Yun, "Anytime-Lidar: Deadline-aware 3D object detection," in *Proc. IEEE Int. Conf. Embed. Real-Time Comput. Syst. Appl. (RTCSA)*, 2022, pp. 31–40.

[12] H. Caesar et al., "nuScenes: A multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 1–11.

[13] "Jetson AGX xavier developer kit." NVIDIA. Accessed: Mar. 30, 2024. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit

[14] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu, "Class-balanced grouping and sampling for point cloud 3D object detection," 2019, *arXiv:1908.09492*.

[15] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/10/3337

[16] M. Boddy and T. L. Dean, "Solving time-dependent planning problems,"'in *Proc. 11th Int. Joint Conf. Artif. Intell.*, 1989, pp. 979–984.

[17] S. Zilberstein, F. Charpillet, and P. Chassaing, "Real-time problem-solving with contract algorithms," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 1999, pp. 1–6.

[18] H. Tang, Z. Liu, X. Li, Y. Lin, and S. Han, "TorchSparse: Efficient point cloud inference engine," in *Proc. Conf. Mach. Learn. Syst. (MLSys)*, 2022, pp. 1–14.

[19] B. Graham and L. van der Maaten, "Submanifold sparse convolutional networks," 2017, *arXiv:1706.01307*.

[20] O. D. Team. "OpenPCDet: An open-source toolbox for 3D object detection from point clouds." 2020. [Online]. Available: https://github.com/open-mmlab/OpenPCDet

[21] T. Schreier, K. Renz, A. Geiger, and K. Chitta, "On Offline evaluation of 3D object detection for autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops (ICCVW)*, 2023, pp. 1–6. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCVW60793.2023.00441

[22] S. Kato et al., "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proc. ACM/IEEE Int. Conf. Cyber Phys. Syst. (ICCPS)*, 2018, pp. 287–296.

[23] "Apollo: Open source autonomous driving." 2017. [Online]. Available: https://github.com/ApolloAuto/apollo

[24] M. Alcon, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Timing of autonomous driving software: Problem analysis and prospects for future solutions," in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, 2020, pp. 1–14.

[25] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Process. Mag.*, vol. 37, no. 4, pp. 50–61, Jul. 2020.

[26] P. Sun et al., "Scalability in perception for autonomous driving: Waymo open dataset," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 1–9.

[27] C. M. G. Shi and R. Li, "PillarNet: Real-time and high-performance pillar-based 3D object detection," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2022, pp. 1–18.

[28] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, "From points to parts: 3D object detection from point cloud with part-aware and part-aggregation network," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 8, pp. 2647–2664, Aug. 2021.

[29] T. Zhao et al., "Ada3D: Exploiting the spatial redundancy with adaptive inference for efficient 3D object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2023, pp. 1–11. [Online]. Available: https://api.semanticscholar.org/CorpusID:259937318

[30] J. Liu, Y. Chen, X. Ye, Z. Tian, X. Tan, and X. Qi, "Spatial pruned sparse convolution for efficient 3D object detection," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2022, pp. 1–14. [Online]. Available: https://openreview.net/forum?id=QqWqFLbllZh

[31] S. Lee and S. Nirjon, "SubFlow: A dynamic induced-subgraph strategy toward real-time DNN inference and training," in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, 2020, pp. 15–29.

[32] S. Heo, S. Cho, Y. Kim, and H. Kim, "Real-time object detection system with multi-path neural networks," in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, 2020, pp. 174–187.

[33] S. Heo, S. Jeong, and H. Kim, "RTScale: Sensitivity-aware adaptive image scaling for real-time object detection," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2022, pp. 1–22.

[34] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2021, pp. 8806–8813.

[35] Y. Hu, S. Liu, T. Abdelzaher, M. Wigness, and P. David, "On exploring image resizing for optimizing criticality-based machine perception," in *Proc. IEEE Int. Conf. Embed. Real-Time Comput. Syst. Appl. (RTCSA)*, 2021, pp. 169–178.

[36] S. Liu et al., "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2020, pp. 319–332.