MURAL: A Multi-Resolution Anytime Framework for LiDAR Object Detection Deep Neural Networks

Ahmet Soyyigit[1], Shuochao Yao[2], and Heechul Yun[3]

[1,3]University of Kansas, Lawrence, KS, {ahmet.soyyigit,heechul.yun}@ku.edu
[2]George Mason University, Fairfax, VA, shuochao@gmu.edu

***Abstract*—Making tradeoffs between execution latency and result utility (i.e. anytime computing) to adapt dynamic operation requirements has been demonstrated to improve the performance of cyber-physical systems. In this work, we focus on the problem of enabling anytime computing for deep neural networks (DNN) that process lidar point clouds as input for 3D object detection. We propose a novel method that unlocks multi-resolution inference, allowing input to be scaled and processed in a dynamically determined resolution to satisfy timing requirements. Importantly, our memory-friendly method requires the deployment of a single DNN model only, without the need of deploying multiple models where each model aims a different input resolution. We further propose a deadline-aware scheduler that selects the highest possible resolution for any given input by accurately predicting the execution time for all possible resolutions at runtime, which is challenging due to the irregularity of input point clouds. Experimental results on the nuScenes autonomous driving dataset demonstrate that our method significantly outperforms existing anytime computing approaches for lidar object detection. By achieving superior detection accuracy under varying time constraints while maintaining deployment simplicity, our work establishes a new state-of-the-art in this domain. The code will be released.**

***Index Terms*—LiDAR, 3D object detection, Anytime computing**

I. Introduction

Autonomous systems critically depend on the accurate detection of surrounding objects in real-time. For this task, many LiDAR-based object detection deep neural networks (DNNs) achieving high accuracy have been proposed in recent years [1]–[4]. However, these state-of-the-art LiDAR object detection DNNs are computationally expensive, making deployment on resource-constrained embedded computing hardware challenging. This challenge is particularly pronounced in systems with strict size, weight, and power (SWaP) constraints, necessitating trade-offs between accuracy and latency.

The required accuracy/latency trade-offs depend not only on the SWaP constraints but also on the dynamic operating environment [5], [6]. For example, in complex and crowded urban environments where objects move slowly, processing input in a fine-grained manner may be desirable to maximize detection accuracy, even if it takes longer. However, in simpler environments with fast-moving objects, such as highways, it may be preferable to process quickly in a coarse-grained manner, as lower processing latency could be more important than high precision and fine-grained details.

Algorithms that can trade off quality and latency are known as *anytime* algorithms in the literature [7], and there has been significant effort in recent years to make deep neural networks that process perceptual input data anytime-capable. For image classification and object detection tasks, "early-exit" architectures have been explored [8]–[11], where additional output layers are integrated at intermediate stages, allowing the DNN to make predictions before reaching the full depth of the model. Criticality-based slicing and scheduling of input [10], [12]–[14] and dynamically scaling image resolution [15]–[17] have been studied to enable anytime processing capabilities in perception DNNs. However, most prior works have focused on vision-based DNNs.

For LiDAR-based object detection tasks, Anytime-Lidar [18] combined the early-exit method with a novel detection head scheduling technique to enable dynamic latency/accuracy tradeoffs for PointPillar [4]. VALO [19] explores a deadline-aware input slicing and scheduling approach that greatly improves the anytime performance, achieving higher accuracy under the same deadline, when applied to the state-of-the-art LiDAR object detection models [2], [20]. However, LiDAR resolution, defined here as the spatial granularity of the input representation such as pillar size in voxel-based detectors [2], [20], [21], remains a largely underexplored scaling factor in the design of anytime detection models. Although adjusting resolution provides an excellent trade-off between detection accuracy and execution time (as shown in Figure 3), the runtime memory requirements grow linearly with the number of supported resolutions, presenting a key challenge for practical deployment.

In this paper, we propose MURAL, a multi-resolution anytime framework for LiDAR 3D object detection DNNs. First, MURAL enables dynamic selection of processing resolution, allowing flexible trade-offs between accuracy and latency while using a single shared set of network weights. This is possible thanks to its multi-resolution architecture enhancement and training methodology (Section IV-B). Second, MURAL can support arbitrary input resolutions, including those not seen during training, by applying a runtime interpolation technique to batch normalization layers (Section IV-C). Third, MURAL incorporates a deadline-aware scheduler that dynamically selects the highest feasible input resolution with a given time constraint, based on accurate execution time predictions for each resolution (Section IV-D).

We apply MURAL on two state-of-the-art 3D LiDAR object

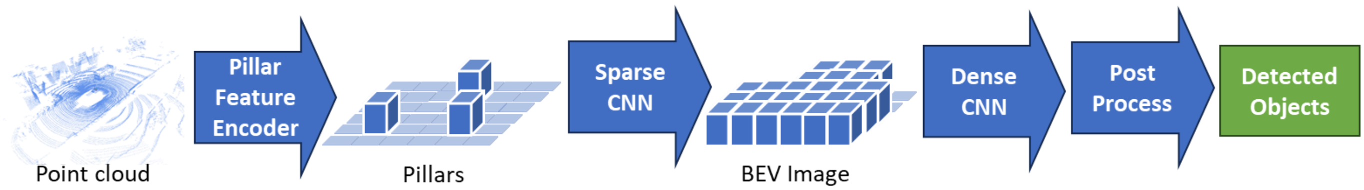

Fig. 1: Architecture of pillar-based LiDAR object detection DNNs.

detection networks, Pillarnet [3], and PointPillars [4], and evaluate their performance in terms of accuracy, latency, and resource demands, compared to the separately trained baseline models for each considered resolution and prior state-of-the-art anytime approach. The results show that MURAL achieves higher detection accuracies than the baselines models for each resolution as well as the prior state-of-the-art anytime approach.

In summary, our contributions are as follows:

- We present the first work exploring runtime resolution scaling in LiDAR object detection DNNs.
- We provide a general framework that can be applicable to any LiDAR 3D object detection networks.
- We achieve superior performance (higher accuracy for a given deadline) compared to the prior state-of-the-art anytime LiDAR approach and non-anytime baseline models.

II. Background

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

A. LiDAR Object Detection DNNs

The LiDAR sensor periodically scans the operated environment and provides a point cloud for each scan. A point cloud P having a set of n points can be defined as:

$$P = \{(x_1, y_1, z_1, i_1), \dots, (x_n, y_n, z_n, i_n)\} \tag{1}$$

where each point has its 3D coordinate in meters and laser return intensity. To accurately detect objects of interest in a point cloud, researchers have proposed using Deep Neural Networks (DNNs) [2]–[4], [22]. To efficiently process point clouds with DNNs, the cubic space S containing the point cloud is divided into a grid G of fixed-size cubical cells. All cells containing one or more points are called voxels. The dimensions of G is calculated as:

$$G = (\frac{X_e - X_s}{V_x}, \frac{Y_e - Y_s}{V_y}, \frac{Z_e - Z_s}{V_z}) \tag{2}$$

where $(X_s, X_e, Y_s, Y_e, Z_s, Z_e)$ define the range of S in LiDAR's coordinate system and (V_x, V_y, V_z) is the voxel dimensions, both in meters. Transforming a point cloud into voxels allows utilizing convolutional neural networks (CNNs) for feature extraction, since the grid that includes the voxels can be considered as a tensor (i.e. multidimensional array).

When the height of the voxels (V_z) is equal to the height of the cubic space $(Z_e - Z_b)$, effectively removing the height dimension of G, the voxels are called instead *pillars*. Many works have proposed using pillars instead of voxels [3], [4] to avoid expensive 3D convolutional layers and thus provide deployment-friendly solutions with minimal or no sacrifice in accuracy.

Figure 1 shows the general architecture of the pillar-based LiDAR object detection DNNs. The pillar feature encoder (PFE) transforms points into pillars defined in coordinate list (COO) format. At this stage, the pillars occupy a very small part of the grid (e.g., 3%-20%), and applying dense convolution on the entire grid can be computationally wasteful. An option to avoid this waste is to keep the pillars in COO format and process them with a sparse CNN [22]. The sparse convolutions mathematically do the same operations as dense convolutions, but on sparse tensors instead of dense. After sparse CNN, the grid is viewed as a bird-eye view image by scattering the sparse output tensor on the dense grid and then processed by a dense CNN. Finally, post-processing operations such as non-maximum suppression are applied to obtain the detection result.

B. Convolution and Batch Normalization

As discussed above, once 3D point cloud input is converted to pillars, the convolutional layers are used to process the pillars, much like processing pixels in images. Convolutional layers apply fixed-size filters on any given input with dimensions NCHW (i.e. batch size, channel, height, width). Because convolutional layers do not enforce a specific height and width for their inputs, they technically can be applied to inputs of any resolution.

Note that CNNs usually include batch normalization (BN) layers after each convolution layer to make model convergence faster and stable. A BN layer is defined as follows:

$$y = \gamma \cdot \frac{x - \mu}{\sigma} + \beta \tag{3}$$

where y is the normalized output, x is the input, μ is the mean, σ is the standard deviation, and γ and β are learnable scale and shift parameters, which are dependent on the statistical distribution of input tensors they process [23].

C. Resolution Scaling of 3D Point Cloud

Given the general architecture of a LiDAR object detection DNN described above, adjusting the size of the pillars can be an effective way to scale the resolution of the detector. Figure 2 illustrates three examples of pillars generated from the same point cloud. Increasing the size of the pillars (V_x, V_y)

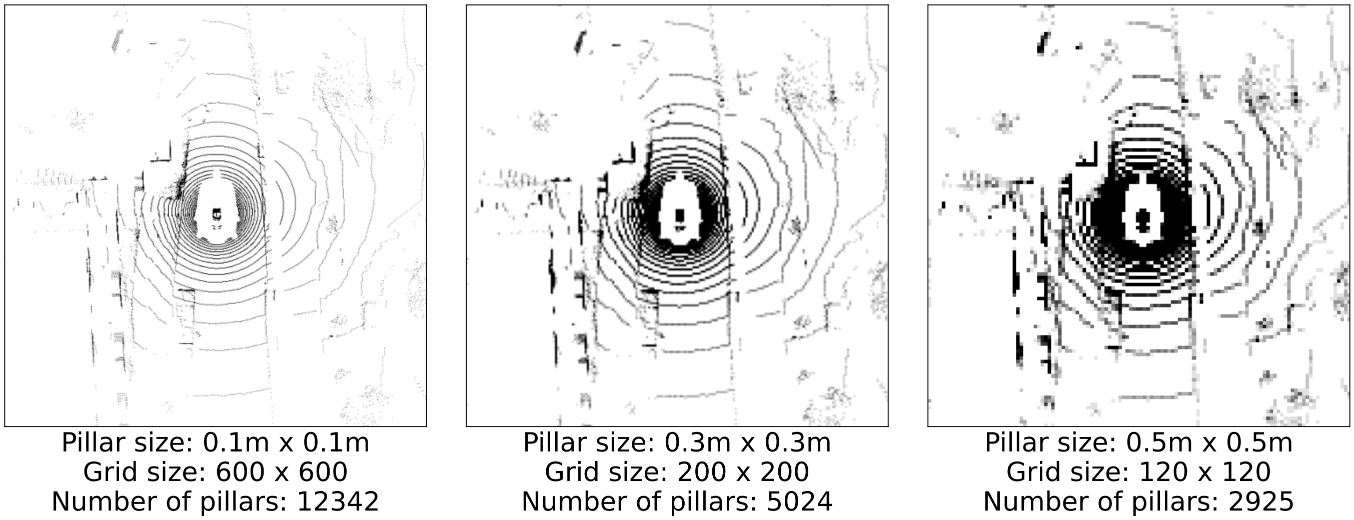

Fig. 2: Bird-eye-views of a LiDAR point cloud transformed into pillars of three sizes. In all cases, the cubical space S that contains the point cloud (thus pillars) is defined by range $(X_s = -30m, X_e = 30m, Y_s = -30m, Y_e = 30m, Z_s = 0m, Z_e = 8m)$. A darker color indicates containing more points in a pillar.

in which the points are encoded reduces the number of pillars and also the height and width dimensions of the grid G. This enables faster processing without the need to change the model architecture. However, using bigger pillars results in a decreased ability to capture fine grained details, similarly to using images of lower resolutions.

III. Motivation

In this section, we explore the feasibility and the challenges of resolution scaling to enable anytime computing capability in LiDAR object detection.

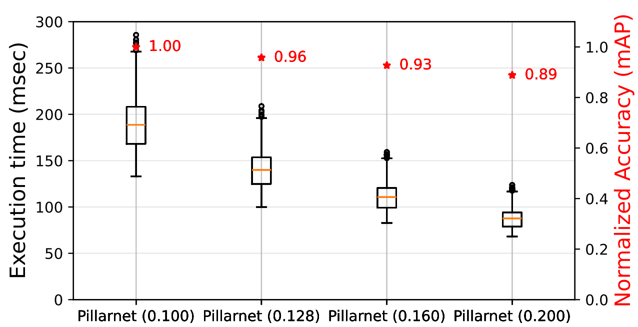

Fig. 3: Execution time (on Jetson AGX Orin at 30W) and accuracy statistics of Pillarnet separately trained with four different pillar sizes $(0.100^2m^2, 0.128^2m^2, 0.160^2m^2, 0.200^2m^2)$. Accuracies are normalized w.r.t. the accuracy of Pillarnet (0.100).

Figure 3 shows the execution time distribution and the average accuracy of Pillarnet [3] LiDAR object detection models, each of which was trained with a different pillar size. As expected, higher resolution leads to higher accuracy but also, on average, to longer execution times. Given this, one simple approach to enable anytime computing is to use multiple models with different resolutions and switch between them depending on the deadline. However, such an approach would require loading multiple DNN models into GPU memory, which may not be feasible on memory-constrained embedded computing platforms.

While it is technically possible to use a resolution (pillar size) different from the specific resolution for which the model was trained, due to the fully-convolutional nature of LiDAR object detection models, it would significantly impact accuracy.

Pillar size (m^2)	**Normalized mAP (%)**
0.100^2	100.0
0.128^2	78.8
0.160^2	41.0
0.200^2	18.0

TABLE I: Accuracy (mAP) of Pillarnet (0.100) when the pillar size used in testing differs than training.

Table I shows the normalized mAP scores of the Pillarnet (0.100) model when used with different resolutions. As can be seen, when the resolution used during testing differs from that used during training, the accuracy drops significantly.

In this work, our goal is to develop a framework that enables a single LiDAR object detection model to have anytime capability by supporting multiple different resolutions without incurring accuracy loss compared to the baseline models that are trained for specific resolutions.

IV. MURAL

In this section, we introduce MURAL, a **MU**lti-**R**esolution **A**nytime **L**iDAR framework, which transforms any pillar-based LiDAR object detection DNN into an anytime algorithm, ensuring that detection results are delivered in a timely manner with the highest possible accuracy

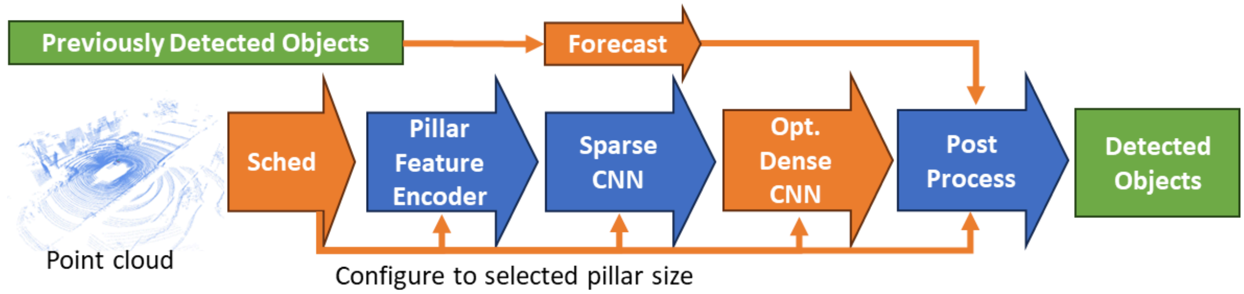

Fig. 4: The architecture of MURAL.

A. Overview

MURAL is designed to enable efficient accuracy–latency trade-offs by dynamically adjusting the pillar size used to encode the input point cloud. Recall that increasing the pillar size reduces the resolution (i.e., the width and height dimensions) of the grid processed by the CNN (Section II-C). By adjusting the pillar size, MURAL ensures that the entire input is processed within the deadline for each invocation of the object detector. To support this capability, MURAL modifies the normalization layers of the target DNN to be resolution-aware and trains the model to adapt to multiple pillar sizes (Section IV-B). After training, MURAL can support additional pillar sizes—beyond those used during training—by interpolating or extrapolating resolution-specific parameters (Section IV-C).

At inference, its scheduler (Section IV-D) takes the input point cloud and predicts the execution time required for each candidate pillar size. It then selects the smallest pillar size (i.e., the highest resolution) that can meet the real-time deadline and configures the rest of the detection pipeline accordingly. MURAL also incorporates two optimizations from [19]. The first eliminates redundant computations in dense CNN layers (Section IV-E), while the second forecasts object positions based on previous detections (Section IV-F). After forecasting, detected and forecasted objects are merged, with priority given to detected objects to improve accuracy.

Figure 4 illustrates the overall runtime architecture of MURAL where blue boxes represent the baseline LiDAR detection models and orange boxes represent our modifications.[YUN: *Isn't that the sparse CNN also should be orange due to the additional BN layers?*]

B. Multi-Resolution Training and Inference

We propose a training scheme for LiDAR object detection DNNs that enables a single model to dynamically decide the pillar size for each input. Importantly, for each pillar size, the accuracy delivered by our model is expected to be comparable to or better than that of separately trained models. In this way, a MURAL-applied DNN can replace multiple DNNs, enabling efficient and memory-friendly trade-offs between accuracy and latency.

In our training scheme, for each point cloud input frame, we perform a separate forward pass for all targeted pillar sizes and accumulate the loss values calculated for each, as follows:

$$\mathcal{L}_{\text{total}} = \sum_{p \in \mathcal{P}} \mathcal{L}(f_\theta(x_s), y) \tag{4}$$

where $\mathcal{L}_{\text{total}}$ is the total accumulated loss, $\mathcal{P}$ is the set of all targeted pillar sizes, $\mathcal{L}$ is the loss function of the baseline DNN, f_θ is the DNN with parameters θ, x_s is the input point cloud, which is encoded and processed into pillars of size p and y is the ground truth.

Then, we apply backpropagation using $\mathcal{L}_{\text{total}}$ and update the model weights θ. This allows the parameters to be updated with gradients accumulated from all resolutions. Although this approach makes the DNN adaptable to $\mathcal{P}$—to a certain degree—we find that the accuracy obtained for each pillar size still falls noticeably short of that achieved by models trained separately.

To address this issue, we introduce separate batch normalization (BN) layers for each input resolution. Our design is inspired by a prior study on image classification [16], which finds that different image resolutions produce distinct statistical distributions that affect the behavior of batch normalization (see Section II-C). They propose using per-resolution BN layers as an effective way to support multiple image resolutions while sharing convolutional weights.

We hypothesize that LiDAR pillars exhibit similarly resolution-dependent distributions, which motivates our design choice of applying per-resolution BN layer to every CNN layer in LiDAR object detection networks.

During training, since we perform forward passes with each pillar size, all BN layers are eventually activated. Thus, backpropagation updates the parameters of all BN layers. At runtime, however, depending on the selected pillar size, we dynamically activate the corresponding normalization layers in the DNN. Note that because each BN layer contains only a few parameters, using separate layers for each resolution does not incur a noticeable memory overhead, while it significantly improve accuracy.

C. Supporting Arbitrary Resolution at Inference

At inference time, restricting ourselves to only the pillar sizes used during training results in coarse-grained la-

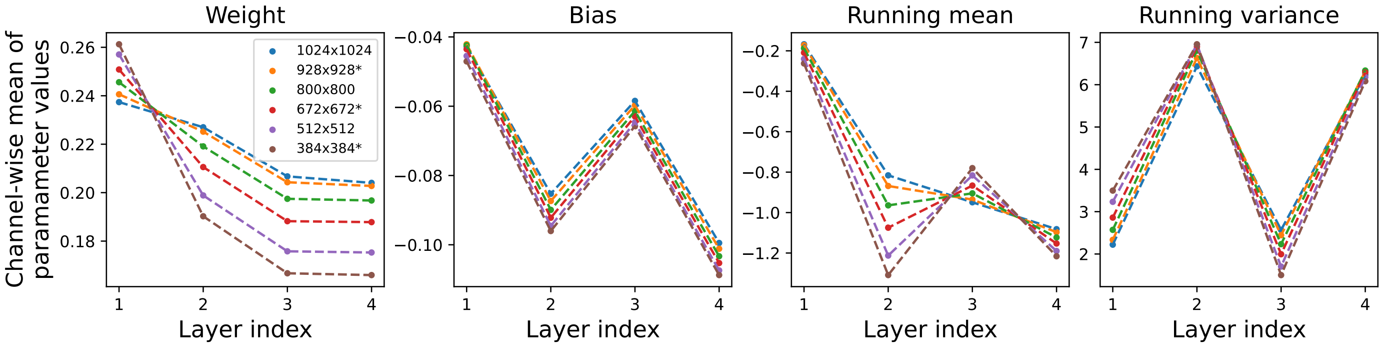

Fig. 5: Channels-wise means of all batch normalization parameters (weight, bias, running mean, running variance) for six grid areas from 1024×1024 to 384×384. The grid areas of the additional pillar sizes are indicated with a (*) in the legend. Their predicted batch normalization parameters are crafted with interpolation/extrapolation.

tency–accuracy trade-offs due to the limited set of options. To provide greater flexibility, we enable the use of arbitrary pillar sizes at runtime by artificially generating batch normalization (BN) layers—without retraining. The parameters of these artificially created BN layers are predicted by interpolating or extrapolating from the parameters of the trained BN layers. Specifically, for each BN parameter (mean, variance, scale γ, and shift β), we model its relationship with the pillar size (or equivalently, the input grid area) as a quadratic function, and apply interpolation or extrapolation independently for each channel. Interpolation is used when the target pillar size lies between two trained sizes, while extrapolation applies when the target size falls outside the range of the trained values.

Figure 5 shows BN layers for six different resolutions, where channel-wise means are visualized to illustrate that the relationship between grid area and parameter values can be modeled independently for each layer. Note that only three of these were obtained through training, while the other three (marked with '*') were generated artificially after training.

This approach allows our model to generalize to input grid areas beyond those seen during training. Empirically, we observe that the accuracy achieved with interpolated pillar sizes falls between the accuracies of the two closest training pillar sizes. Additionally, we support extrapolation to pillar sizes larger than the maximum used in training, enabling MURAL to meet strict latency requirements.

D. *Deadline-aware Resolution Scheduling*

To maximize detection accuracy within a dynamically given deadline, we propose scheduling the highest input resolution (i.e., the smallest pillar size) that meets the deadline. This requires accurately predicting the model's latency for multiple pillar sizes, which is a non-trivial task due to the highly varying latencies.

To tackle the time prediction challenge, we break down the latency L of a LiDAR object detection model into four parts:

$$L = L_{PFE} + L_{SC} + L_{DC} + L_{PP} \tag{5}$$

where the four components indicate the latency of the pillar feature encoder, sparse CNN, dense CNN, and post-processing, respectively.

Figure 6-a shows the latencies of these four components for a representative LiDAR object detection DNN [3]. Note, first, that L_{PP} and L_{DC} are highly predictable. Therefore, it is viable to use their 99th percentile values, acquired from offline benchmarking for each input resolution, to predict them.

For L_{PFE}, shown in Figure 6-b, there is a strong correlation between the number of input points and the latency without significant variance. Thus, we use a simple quadratic equation based regression to predict it.

Finally, L_{SC} exhibits high variation, as illustrated in Figure 6-c, even when processing the same number of input pillars. This variability makes it difficult to predict execution time using either a fixed worst-case estimate or a simple quadratic equation as used for L_{PFE}.

This variation occurs because each sparse convolution inside the sparse CNN produces a different number of output pillars for the same number of input pillars. The coordinates of the output pillars (representing non-zero elements in the grid) depend not only on the count but also on the specific spatial distribution (i.e. coordinates) of the input pillars, as shown in Figure 7. This spatial dependency creates a cascade effect through the sparse CNN, causing the number of active pillars to dynamically vary between layers - the primary factor behind the variability of execution time [19].

Although the execution time of any individual sparse convolution can be accurately modeled with a quadratic equation if its input pillar count is known, the challenge lies in predicting these counts before execution. Our previous work [19] used a history-based approach, assuming temporal consistency between consecutive LiDAR frames. However, this assumption breaks down in highly dynamic environments.

In this work, we propose a more robust method that estimates the input pillar counts for all sparse convolutions without executing the actual sparse CNN. Our approach uses lightweight max pooling operations configured to mimic the sparse convolution layers. The key insight is that for a given input, the output pillar coordinates produced by a convolution

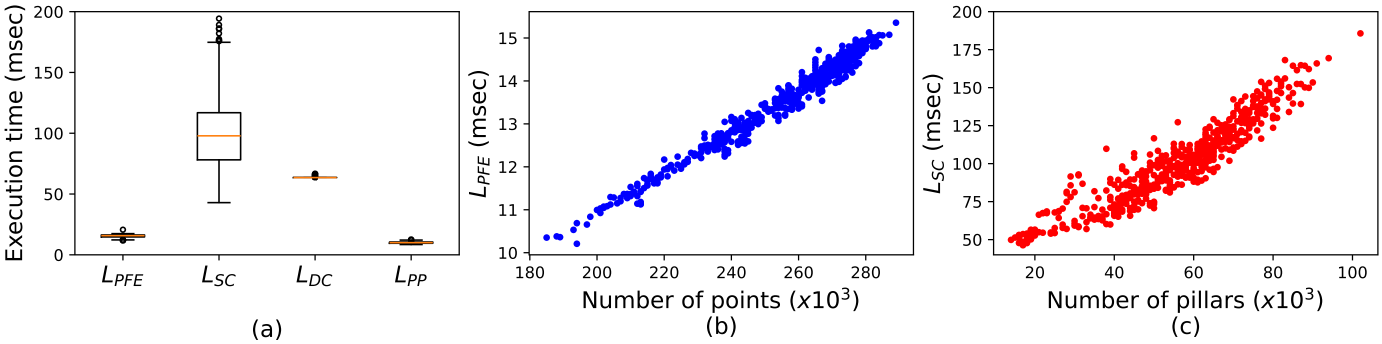

Fig. 6: (a) Component-wise execution timing of the Pillarnet (pillar size is $0.100^2 m^2$). (b) PFE latency of the same Pillarnet with respect to its input. (c) Sparse CNN latency of the same Pillarnet with respect to its input.

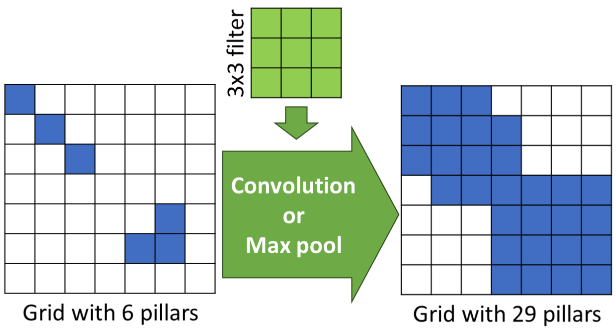

Fig. 7: Example of convolution and maximum pooling producing same nonzero coordinates, assuming all elements in the grid are greater or equal to zero when max pooling is applied.

(whether sparse or dense) can be accurately reproduced by an appropriately configured max pooling operation (using the same kernel size, stride, and padding as the convolution it mimics). After calculating the input pillar counts of all sparse convolutions layers with max pooling operations, we predict L_{SC} by mapping these pillar counts to execution times with quadratic equations per layer and take their sum. Note that all pillar count calculations are done for all pillar sizes considered for scheduling.

After predicting the execution times, the MURAL's scheduler picks the smallest pillar size for a given input which can meet the deadline. MURAL dynamically reconfigures the DNN to accommodate the selected pillar size by following the below steps:

1) PFE is configured to encode the pillars to selected pillar size when transforming points to pillars.
2) The normalization layers of the selected pillar size is activated in PFE, sparse CNN, and dense CNN.
3) Post-processing is informed of the resolution change to make processing of output tensors correct since the output tensor sizes change with respect to the input.

E. Dense CNN Optimizations

The point cloud from LiDAR may occupy a smaller area in the BEV than the cuboidal space defined by its range. As a result, large portions of the grid, especially at the edges, can be empty. We crop these empty regions to speed up the dense CNN without sacrificing accuracy, as implemented in our prior work [19]. Additionally, dense convolutions that infer object attributes (e.g., size, velocity) may perform redundant computations across the entire grid, including empty regions. To avoid this, we apply an optimization from [19] that limits inference to regions with detected objects, reducing latency while maintaining accuracy. This optimization is also incorporated into MURAL.

F. Forecasting

The forecasting process involves predicting the current position of objects detected in previous frames using their inferred velocity and ego-vehicle localization information. In our prior work [19], we used input data scheduling to make latency–accuracy trade-offs. Since the approach skips processing a portion of the data, it employed forecasting of past detection results to compensate for the skipped information. Interestingly, we found forecasting to be beneficial for MURAL, even though no input data is skipped. It allows objects missed in the current frame (e.g., due to occlusion) but detected in the previous frame to be continuously tracked. Therefore, we incorporate forecasting into MURAL.

V. Evaluation

We extended OpenPCDet [24], an open-source LiDAR object detection framework that supports state-of-the-art methods, to implement MURAL. Our primary evaluation uses PillarNet [3], a leading pillar-based DNN. To demonstrate MURAL's general applicability, we also present results on PointPillars [4] with CenterHead [2] attached. Unlike PillarNet, PointPillars does not employ a sparse CNN, which simplifies scheduling because latency prediction becomes more straightforward.

For comparison, we evaluated MURAL against VALO [19], a state-of-the-art anytime LiDAR object detection framework that achieves its anytime capability through input data slicing

	Jetson AGX Xavier	Jetson AGX Orin
CPU	8-core NVIDIA Carmel	8-core Arm Cortex-A78AE
RAM	16 GB	32 GB
OS	Ubuntu 20.04	Ubuntu 22.04
Software	Jetpack 5.1	Jetpack 6.0

TABLE II: Experiment platforms

and scheduling. In VALO, its baseline model is configured to use the smallest pillar size to maximize accuracy.

For training and testing of the models, we use the nuScenes [25] autonomous driving dataset and report detection accuracy using the mean average precision (mAP) metric. Training is performed with the entire training split of nuScenes containing 700 distinct scenes, each containing a 20-second LiDAR scan sequence sampled at 50-millisecond intervals. For runtime evaluation, we use 75 distinct scenes from the nuScenes validation split, and process all annotated frames in each sequence—spaced 250 milliseconds apart—one by one. We repeated this process under different deadline constraints for each tested model. During both training and testing, we merge the 10 most recent LiDAR scans for each input. This technique is commonly used to enhance accuracy and to enable object velocity estimation using DNNs [25].

To assess detection timeliness, we evaluate under varying deadline constraints. Our testing methodology maintains a buffer of the most recent successful detection results, updating it whenever a method meets its deadline. When a deadline is missed, we discard the late output and instead use the buffered results, simulating job abortion.

We conducted performance evaluations on two platforms: NVIDIA Jetson AGX Xavier and NVIDIA Jetson AGX Orin, both configured with a 30W power profile. Details of the platforms are provided in Table II. On both devices, we dedicated six CPU cores and all available GPU resources exclusively to the method under test.

Our evaluation results are organized into six subsections: (1) details of MURAL's training; (2) MURAL's performance on Pillarnet; (3) MURAL's performance on PointPillars; (4) an ablation study of MURAL's components; (5) an analysis of the scheduler's time prediction errors; and (6) overhead analysis. In each section (except the first), we normalize the detection scores (mAP) of all evaluated methods relative to the highest score obtained in that section.

A. Training Results

Using the nuScenes dataset, we first trained the baseline Pillarnet three times, each with a different pillar size. We then compared MURAL's performance at different resolutions with the corresponding baseline Pillarnet models. Note that in this experiment, we assumed no deadline violations to isolate the accuracy implications of MURAL.

Table III shows the results. Note that MURAL maintains comparable accuracy for the smallest (0.100^2) and largest (0.200^2) pillar sizes, while achieving higher accuracy for the medium pillar size (0.128^2). This demonstrates that MURAL, despite being a single model supporting multiple resolutions, achieves comparable or better accuracy than individual models trained for specific resolutions. We posit that the improved accuracy stems from the regularization effect of multi-resolution training, as also suggested in [16] for multi-resolution image classification.

Pillar size (m^2)	**Pillarnet baseline**	**MURAL**
0.100^2	0.590	0.591 (+0.001)
0.128^2	0.565	0.583 (+0.018)
0.200^2	0.524	0.519 (-0.005)

TABLE III: Accuracy in mAP of baseline Pillarnet and the MURAL-applied version. Numbers in parenthesis indicate the differences with respect to the baseline.

In the next experiment, to evaluate the effectiveness of arbitrary resolution support described in Section IV-C, we introduce several non-trained resolutions and evaluate their mAP scores.

Table IV shows the results. Note that the blue color represents the additional resolutions introduced post-training. The results show that these arbitrary resolutions, enabled by interpolated BN layers, achieve good accuracy, falling between the neighboring trained resolutions.

Pillar size (m^2)	**Grid area**	**mAP**
0.100^2	1024^2	0.591
0.109^2	928^2	0.591
0.128^2	800^2	0.583
0.151^2	672^2	0.558
0.200^2	512^2	0.519
0.263^2	384^2	0.408

TABLE IV: MURAL on Pillarnet with post-training introduced pillar sizes (blue).

B. MURAL on Pillarnet

Figures 8-a and 8-b illustrate how detection accuracy changes with respect to the deadline, with MURAL outperforming the baselines on both platforms. Due to its dynamic resolution scheduling, MURAL can select from a wide range of resolutions to meet a given deadline, maximizing accuracy within any given time constraint, as shown in Figure 9. However, the Pillarnet baseline models can only provide predictions for a much narrower range of deadlines, as they lack anytime computing capability, resulting in lower accuracies. In contrast, VALO [19] has anytime capability and can achieve higher accuracies than the baselines across a wider range of deadlines. Nevertheless, its data scheduling approach is less effective than MURAL's dynamic resolution scaling. The primary reason for MURAL's superior performance over VALO is that VALO processes only a small subset of the input data for tight deadlines, making it more dependent on forecasted detections. MURAL, on the other hand, processes the entire input frame, albeit at a lower resolution, regardless of the deadline. As a result, MURAL achieves higher accuracies than both VALO and the separately trained baseline models.

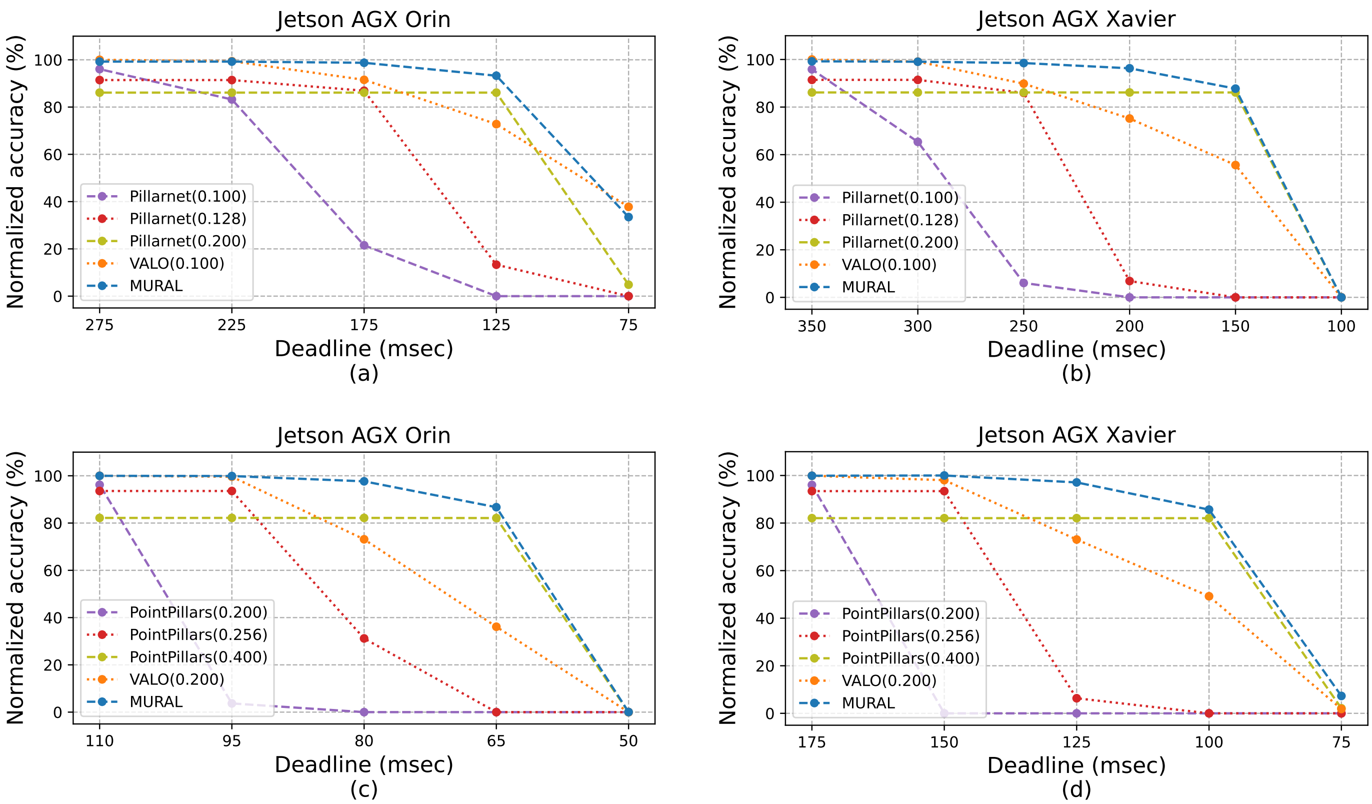

Fig. 8: (a,b) Pillarnet and (c,d) PointPillars experiments on both evaluation platforms. In each plot figure, MURAL and VALO were applied on the baseline it is compared with.

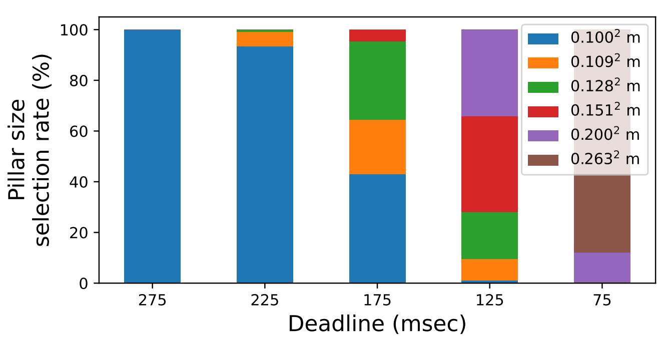

Fig. 9: Pillar size selection rates of MURAL during Pillarnet experiment on Orin.

C. MURAL on PointPillars

We also evaluate MURAL using PointPillars [4], comparing its performance against multiple baseline PointPillars models and a VALO [19] version applied to PointPillars. The MURAL model was trained to support all pillar sizes used in the baselines, along with four additional resolutions introduced after training via BN interpolation (see Section IV-C).

Figures 8-c and 8-d present the results. Similarly to Pillarnet, MURAL maintains better or comparable accuracy across all the tested deadlines compared to the PointPillars baselines and the VALO-applied model.

Overall, the results show that MURAL is generalizable to multiple DNNs and efficient across different computing platforms, establishing it as the new state-of-the-art anytime LiDAR object detection method.

D. Ablation study

We also performed an ablation study of MURAL on Pillarnet by comparing the following methods:

- **DS-PSI-DCO-FRC**: MURAL with all its four components: dynamic scheduling (Section IV-D), introducing post-training pillar sizes (Section IV-C), dense convolution optimization (Section IV-E), and forecasting (Section IV-F).
- **DS-PSI-DCO**: MURAL without forecasting.
- **DS-PSI**: MURAL without forecasting and dense CNN optimizations.
- **DS**: MURAL with dynamic scheduling only.
- **SS**: MURAL with static scheduler that choose the highest possible input resolution by considering the worst-case execution time (WCET) of all resolutions. These WCET's were obtained by offline benchmarking.

Table V show the results in which none of the compared methods missed any deadlines shown in the table. Using dynamic scheduling (DS) improves performance over static

MURAL variant	Deadlines (ms)		
	225	175	125
SS	96.17	85.60	85.60
DS	96.70	95.08	87.68
DS-PSI	97.03	96.14	89.66
DS-PSI-DCO	96.93	96.46	91.28
DS-PSI-DCO-FRC	100.00	99.49	93.98

TABLE V: Normalized accuracy results of MURAL (on Pillarnet) variants created for the purpose of ablation study. DS-PSI-DCO-FRC is the actual MURAL.

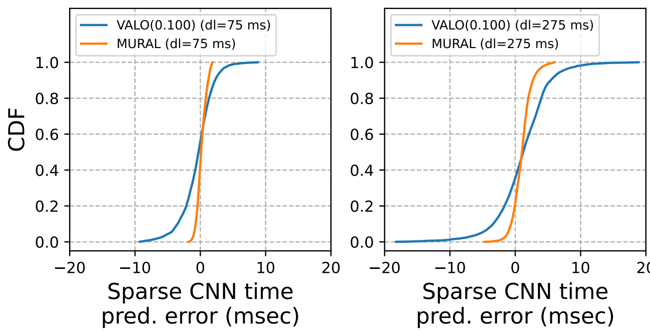

Fig. 10: Time prediction errors of VALO and MURAL applied Pillarnet on Jetson AGX Orin. The errors were calculated by subtracting actual time from the predicted time.

scheduling (SS) because its more accurate execution time prediction. Introducing new pillar sizes (DS-PSI) gives more flexibility to utilize the time until the deadline, improving accuracy. Adding dense convolution optimization (DS-PSI-DCO) benefits more as the deadline becomes tighter, since it allows choosing higher resolutions more without sacrificing accuracy. Finally, forecasting previous detection result to the time of ongoing detection notably improves performance (DS-PSI-DCO-FRC), even though MURAL processes the entire input regardless of deadline. This benefit comes from allowing occluded or missed objects to be detected again in case they were detected in the previous frame.

E. Time Prediction Error

In this experiment, we investigate the time prediction error of our method's scheduler and compare it with that of VALO [19]. In particular, we focus on the sparse CNN, as time prediction for the dense CNN and post-processing stages is straightforward and works the same in both MURAL and VALO. Figure 10 illustrates the time prediction errors as cumulative distribution functions.

As shown in the figure, MURAL outperforms VALO by accurately predicting the number of input pillars for each sparse convolution executed in the sparse CNN. This is achieved by efficiently mimicking all sparse convolutions through max-pooling operations. In contrast, VALO assumes the number of input pillars to be the same for each layer, based on the most recent pillar counts observed in the past.

However, this assumption does not hold in highly dynamic operating environments, where the 3D structure of consecutive LiDAR scans can differ significantly. As a result, VALO's history-based time prediction method is less effective than MURAL's approach.

MURAL variant	Applied baseline	
	Pillarnet	PointPillars
SS	0.31	0.13
DS	3.23	0.53
DS-PSI	5.47	1.17
DS-PSI-DCO	6.22	2.01
DS-PSI-DCO-FRC	6.24	1.97

TABLE VI: Average scheduling overhead (milliseconds) of MURAL variants on Jetson AGX Orin.

	Pillarnet	PointPillars
Baseline	61.003×5	23.956×5
MURAL	61.378	24.259

TABLE VII: Memory in MiB (megabytes) needed to store DNN parameters in 32-bit floating-point format.

F. Time and Memory Overhead Analysis

Table VI shows the average scheduling overhead of MURAL variants, measured on the Jetson AGX Orin. For the static scheduler (SS), the overhead is negligible since we simply consider the worst-case execution times acquired from offline benchmarking. Using the dynamic scheduler (DS) notably increases the overhead of MURAL on Pillarnet, as sparse convolutions of the sparse CNN are mimicked for all resolutions. However, the overhead increases only slightly for PointPillars, as there is no sparse CNN. When we introduce post-training pillar sizes (DS-PSI), the scheduling overhead increases with respect to the number of resolutions that need to be considered for scheduling. Adding dense convolution optimization (DCO) requires determining the empty parts of the input scene in BEV, which increases the scheduling overhead. However, cropping these empty parts accelerates the dense CNN, compensating for the overhead. Enabling forecasting (DS-PSI-DCO-FRC) incurs no significant overhead, as it occurs in parallel on the CPU while the DNN layers execute on the GPU.

Finally, Table VII shows the memory overhead of MURAL compared to using multiple Pillarnet baseline models with different resolutions. Note that MURAL, despite supporting five different resolutions, uses almost the same number of parameters as a single baseline model that supports only one resolution. This is because MURAL's memory overhead for supporting a new resolution is limited to the parameters for per-resolution BN layers, which are minimal compared to the weights of the convolution layers. As a result, MURAL's memory overhead increases only slightly as a function of the number of resolutions it supports, whereas the memory overhead of the baseline models increases multiplicatively with the number of supported resolutions.

VI. Related Work

[YUN: *need revision*] Cyber-physical system software necessitates prompt execution for safety and efficiency. Traditional approaches employing fixed deadlines established during design [26], [27] lack adaptability to varying execution time requirements [28]. To address this, Gog et al. [6] proposed dynamically adjusting deadlines based on driving conditions to enhance vehicle performance and safety.

”Anytime perception” in neural networks, which balances time and accuracy to meet specific deadlines, has been extensively studied. Lee et al. [29] prioritized critical neurons by deactivating others to reduce processing time. Kim et al. [30] achieved this through incremental layer addition and early exits in image classification networks. Yao et al. [9] and Bateni et al. [31] explored scheduling multiple DNN tasks using imprecise computation with early exits and per-layer approximation, respectively. These methods primarily focused on image classification, which differs significantly from complex object detection tasks.

For object detection under deadline constraints, Kuhse et al. [8] analyzed early exit strategies. Heo et al. [32] developed a multipath architecture for anytime perception. Hu et al. [12] proposed adaptive resolution reduction in less critical scene areas. Lie et al. [10], [13] segmented image frames into criticality-based sub-regions, utilizing LiDAR data for priority processing. Kang et al. [14] employed a split-and-merge technique, processing critical image regions at high resolution and non-critical regions at low resolution. Gog et al. [33] suggested dynamic DNN switching. Heo et al. [15] introduced adaptive image scaling based on operational environment, training a single DNN for multi-resolution processing. However, their evaluation lacked comparison with single-resolution baseline models. These object detection efforts primarily address 2D vision, neglecting the unique characteristics of 3D LiDAR point cloud object detection.

LiDAR object detection is crucial for autonomous driving [1]. With large-scale datasets [25], [34], research has focused on both accuracy and latency reduction [2]–[4], [20], [35]–[37]. While these models perform well on high-end hardware, edge deployment remains challenging [19], and they lack deadline-adaptive execution.

Soyyigit et al. [18] proposed Anytime-LiDAR, utilizing early exits and detection head scheduling for non-sparse CNN models. However, its effectiveness is limited with modern sparse CNN models [2], [3], [20]. Subsequently, VALO [19] introduced a data-scheduling approach for anytime computing, maximizing input processing within deadlines and forecasting skipped data for accuracy. VALO, while flexible, suffers from accuracy degradation at tight deadlines due to partial input processing. Yuhang et al. [38] explored multi-modal BEV detection, dynamically skipping camera processing and LiDAR scans. However, their data scheduling is not directly applicable to single-modal models.

Reducing input resolution offers significant latency reduction with minimal accuracy loss. However, dynamic resolution adjustment in a single DNN, maintaining or improving upon single-resolution baseline accuracy, is challenging. Wang et al. [16] used resolution-sensitive batch normalization and ensemble distillation for image classification. Zhu et al. [17] incorporated a resolution predictor network. Chin et al. [39] employed a resolution predictor for video object detection, leveraging temporal consistency. Unlike these works, our research addresses dynamic resolution inference for real-time LiDAR object detection.”

VII. Conclusion

This paper presented MURAL, a multi-resolution anytime framework for LiDAR 3D object detection that balances detection accuracy and processing latency through dynamic resolution scaling. Our approach combines multi-resolution training with shared weights, batch normalization parameter interpolation for arbitrary resolution support, and deadline-aware scheduling to provide a memory-efficient solution for anytime LiDAR object detection.

Experiments with Pillarnet and PointPillars demonstrate that MURAL achieves higher detection accuracies across various deadlines compared to both baseline models and prior state-of-the-art approaches, particularly under tight deadline constraints. By eliminating the need to store multiple model variants, MURAL offers the state-of-the-art solution for resource-constrained embedded platforms with significant SWaP limitations.

References

[1] Y. Li and J. Ibanez-Guzman, “Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems,” *IEEE Signal Processing Magazine*, vol. 37, no. 4, 2020.

[2] T. Yin, X. Zhou, and P. Krähenbühl, “Center-based 3d object detection and tracking,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[3] C. M. Guangsheng Shi, Ruifeng Li, “Pillarnet: Real-time and high-performance pillar-based 3d object detection,” in *European Conference on Computer Vision (ECCV)*, 2022.

[4] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “Pointpillars: Fast encoders for object detection from point clouds,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[5] G.-E. Sela, I. Gog, J. Wong, K. K. Agrawal, X. Mo, S. Kalra, P. Schafhalter, E. Leong, X. Wang, B. Balaji, J. Gonzalez, and I. Stoica, “Context-aware streaming perception in dynamic environments,” Aug. 2022. arXiv:2208.07479 [cs].

[6] I. Gog, S. Kalra, P. Schafhalter, J. E. Gonzalez, and I. Stoica, “D3: a dynamic deadline-driven approach for building autonomous vehicles,” in *European Conference on Computer Systems (EuroSys)*, 2022.

[7] M. Boddy and T. L. Dean, *Solving time-dependent planning problems*. Brown University, Department of Computer Science, 1989.

[8] D. Kuhse, H. Teper, S. Buschjäger, C.-Y. Wang, and J.-J. Chen, “You only look once at anytime (anytimeyolo): Analysis and optimization of early-exits for object-detection,” 2025.

[9] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher, “Scheduling real-time deep learning services as imprecise computations,” in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2020.

[10] S. Liu, S. Yao, X. Fu, H. Shao, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, “Real-time task scheduling for machine perception in intelligent cyber-physical systems,” *IEEE Transactions on Computers*, vol. 71, no. 8, 2022.

[11] J.-E. Kim, R. Bradford, M.-K. Yoon, and Z. Shao, “Abc: Abstract prediction before concreteness,” in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020.

[12] Y. Hu, S. Liu, T. Abdelzaher, M. Wigness, and P. David, “On exploring image resizing for optimizing criticality-based machine perception,” in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021.

[13] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[14] W. Kang, S. Chung, J. Y. Kim, Y. Lee, K. Lee, J. Lee, K. G. Shin, and H. S. Chwa, "Dnn-sam: Split-and-merge dnn execution for real-time object detection," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 160–172, 2022.

[15] S. Heo, S. Jeong, and H. Kim, "Rtscale: Sensitivity-aware adaptive image scaling for real-time object detection," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2022.

[16] Y. Wang, F. Sun, D. Li, and A. Yao, "Resolution switchable networks for runtime efficient image recognition," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV*, (Berlin, Heidelberg), p. 533–549, Springer-Verlag, 2020.

[17] M. Zhu, K. Han, E. Wu, Q. Zhang, Y. Nie, Z. Lan, and Y. Wang, "Dynamic resolution network," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, (Red Hook, NY, USA), Curran Associates Inc., 2021.

[18] A. Soyyigit, S. Yao, and H. Yun, "Anytime-lidar: Deadline-aware 3d object detection," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2022.

[19] A. Soyyigit, S. Yao, and H. Yun, "Valo: A versatile anytime framework for lidar-based object detection deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 4045–4056, 2024.

[20] Y. Chen, J. Liu, X. Zhang, X. Qi, and J. Jia, "Voxelnext: Fully sparse voxelnet for 3d object detection and tracking," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[21] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12697–12705, 2019.

[22] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, 2018.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 448–456, PMLR, 2015.

[24] O. D. Team, "Openpcdet: An open-source toolbox for 3d object detection from point clouds." https://github.com/open-mmlab/OpenPCDet, 2020.

[25] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[26] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2018.

[27] "Baidu Apollo team (2017), Apollo: Open Source Autonomous Driving." https://github.com/ApolloAuto/apollo.

[28] 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 *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[29] S. Lee and S. Nirjon, "Subflow: A dynamic induced-subgraph strategy toward real-time dnn inference and training," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[30] J.-E. Kim, R. Bradford, and Z. Shao, "Anytimenet: Controlling time-quality tradeoffs in deep neural network architectures," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020.

[31] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *IEEE Real-Time Systems Symposium (RTSS)*, 2018.

[32] S. Heo, S. Cho, Y. Kim, and H. Kim, "Real-time object detection system with multi-path neural networks," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[33] 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 *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[34] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[35] 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 Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 8, 2020.

[36] T. Zhao, X. Ning, K. Hong, Z. Qiu, P. Lu, Y. Zhao, L. Zhang, L. Zhou, G. Dai, H. Yang, and Y. Wang, "Ada3d : Exploiting the spatial redundancy with adaptive inference for efficient 3d object detection," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[37] J. Liu, Y. Chen, X. Ye, Z. Tian, X. Tan, and X. QI, "Spatial pruned sparse convolution for efficient 3d object detection," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[38] Y. Xu, Z. Liu, X. Fu, S. Liu, F. Wu, and G. Chen, "Flex: Adaptive task batch scheduling with elastic fusion in multi-modal multi-view machine perception," in *2024 IEEE Real-Time Systems Symposium (RTSS)*, pp. 294–307, 2024.

[39] T.-W. Chin, R. Ding, and D. Marculescu, "Adascale: Towards real-time video object detection using adaptive scaling," in *Proceedings of Machine Learning and Systems 2019*, pp. 431–441, 2019.