Real-Time LiDAR Object Detection on Edge Devices via Adaptive Cropping

Simone Mentasti(✉), Luca Brembilla, and Matteo Matteucci

Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy
{simone.mentasti,luca.brembilla,matteo.matteucci}@polimi.it

Abstract. 3D object detection from LiDAR is a crucial task in autonomous driving and mobile robotics, providing precise 3D bounding boxes of obstacles. State-of-the-art detectors achieve high accuracy but at the cost of processing entire point clouds with heavy neural networks, which makes real-time deployment on resource-constrained edge devices and low-power robots challenging. In this paper, we propose an adaptive cropping strategy for real-time processing of LiDAR data, which significantly reduces computation by focusing only on regions of interest across frames while maintaining detection accuracy. Our method leverages bounding box predictions and vehicle state estimates from each frame to crop the subsequent frame's point cloud to likely object regions, thereby significantly reducing the data volume processed. We integrate this pipeline with modern one-stage and two-stage LiDAR detectors and evaluate on both a custom roadside LiDAR dataset and the public nuScenes dataset. Experimental results show that our approach can process as little as 50% of the points with only a minor mAP decrease for vehicles and pedestrians. On an NVIDIA Jetson AGX Orin edge device, the pipeline significantly improves inference speed, roughly doubling the system FPS. These results demonstrate the feasibility of efficient real-time 3D perception on low-power systems like mobile robots and edge devices.

Keywords: Road User Detection · Autonomous Vehicle · Pointcloud Segmentation

1 Introduction

Autonomous vehicles and mobile robots employ perception pipelines that include 3D LiDAR object detection to locate surrounding objects. LiDAR sensors provide rich 3D point clouds from which detectors predict objects' 3D bounding boxes and classes. Leading LiDAR detection models have achieved impressive accuracy on benchmarks like nuScenes, Waymo Open Dataset, and KITTI. However, these models typically run on powerful GPUs; deploying them on embedded edge devices (e.g., autonomous vehicles, mobile robots, or roadside units) is difficult due to strict processing power and latency constraints. Moreover,

E. Rodolà et al. (Eds.): ICIAP 2025 Workshops, LNCS 16170, pp. 349–359, 2026.
https://doi.org/10.1007/978-3-032-11381-8_30

real-time edge deployment of LiDAR detection is increasingly important with emerging vehicle-to-everything (V2X) applications. For instance, roadside units (RSUs) equipped with LiDAR can detect vehicles or pedestrians in intersection blind spots and share this information with nearby cars (vehicle-to-infrastructure communication). Similarly, a fleet of robots or a group of autonomous vehicles could share perception data. In all these cases, the edge processors often have limited hardware, so efficiency is critical. Traditional LiDAR detectors typically process each point cloud frame independently and in its entirety, which is wasteful and introduces unnecessary latency in continuous data streams. In this work, we address the above challenge by exploiting temporal continuity between consecutive LiDAR frames. We propose an adaptive frame-to-frame point cloud cropping strategy that leverages the detector's previous output to reduce the data processed in the next frame. The core idea behind this approach is that objects move relatively little between successive LiDAR frames if we consider the high frame rate of the sensor. Therefore, regions containing objects at time t will likely include the same objects at time $t + \Delta t$. By focusing computation only on those regions and ignoring other parts of the cloud until necessary, we skip redundant processing of empty or unchanging areas.

We periodically allow a complete point cloud to be processed to discover new objects or recover from any errors, yielding a tunable trade-off between accuracy and speed. We integrate the proposed cropping pipeline with two state-of-the-art LiDAR detectors embodying different paradigms: a one-stage voxel-based network and a two-stage voxel+point network. We evaluate our approach on a new custom LiDAR dataset collected from a fixed roadside sensor, as well as on the standard nuScenes dataset, to validate its generality across both infrastructure-mounted and vehicle-mounted scenarios, where the sensor also moves within the environment. Results show our method can achieve significant speed-ups. In particular, processing only every 2nd or 3rd frame fully and cropping the others yields almost half points processed with mAP drops under 2% for key classes. On an NVIDIA Jetson AGX Orin, our pipeline sustains real-time throughput and even doubles the FPS of a heavy two-stage detector in low-power mode with negligible loss in accuracy. In summary, our contributions are:

- an adaptive multi-frame point cloud cropping strategy to accelerate 3D detection on streaming data.
- an implementation and integration of this strategy with modern LiDAR detectors in both infrastructure (roadside) and mobile (vehicle/robot) settings.
- a thorough evaluation on public and custom data, demonstrating the accuracy vs. computation trade-offs and real-time performance on an embedded edge device.

In this work, we show that efficient 3D perception at the edge is feasible, paving the way for practical V2I-assisted driving and coordinated multi-robot sensing.

2 Related Work

2.1 LiDAR-Based 3D Object Detection

Research in 3D object detection from LiDAR has produced a variety of model architectures. Early approaches operated directly on point sets with point-based networks such as PointNet and PointNet++, which learn point-wise features and aggregate them hierarchically. Voxel-based methods convert the point cloud into a discretized voxel grid for efficient sparse convolution processing. SECOND (Sparsely Embedded Convolutional Detection) [7] is a well-known one-stage voxel-based detector that uses 3D sparse convolution and a bird's-eye view (BEV) projection to detect objects in a single shot. Two-stage detectors, like PV-RCNN (Point-Voxel R-CNN) [5], combine voxel and point-based processing: they first generate coarse region proposals from voxel features, then refine them by pooling point-wise features, achieving higher accuracy at the cost of more computation. Recent advances such as VoxelNeXt [2] further improve voxel-based detection with fully sparse convolutional backbones and optimized feature encoding, achieving state-of-the-art accuracy on benchmarks. Finally, recent efforts have attempted to leverage temporal consistency for improved detection or tracking. CenterPoint [8] proposed a tracking extension that fuses detections across frames. FlowNet3D [4] addresses scene flow estimation, which can inform object motion between frames.

While these models attain high accuracy, they usually assume a static scenario where each point cloud is processed independently, maximizing accuracy per frame. They often neglect frame-to-frame temporal information and are benchmarked on high-end GPUs, without consideration for runtime on low-power devices. In real-world driving, however, point clouds arrive as a continuous stream (e.g., 10–20 Hz) and nearby frames are highly correlated. Processing every frame fully can be inefficient, especially for edge hardware with limited compute capabilities. Recently, there has been growing interest in efficient LiDAR perception, for example, methods that downsample points, use smaller input regions, or compress features to speed up inference, but these typically involve trade-offs that reduce accuracy. Our work contributes to this area by explicitly leveraging temporal consistency to avoid redundant computation, thereby accelerating inference with minimal accuracy loss.

2.2 Datasets and Edge Deployment

Large-scale datasets like KITTI [3], nuScenes [1], and Waymo Open Dataset [6] have driven progress in LiDAR detection. KITTI, collected from a car-mounted LiDAR, was long the standard benchmark, though it covers limited scenarios (only front-facing sensor and a few object classes). NuScenes expanded this with 360-degree coverage and a wider variety of object classes, and introduced an evaluation that includes a tracking metric. Waymo's dataset further increased scale (hundreds of scenes, millions of objects) and diversity. These datasets focus on on-vehicle sensors; by contrast, our custom dataset features a LiDAR mounted

on an infrastructure (roadside) unit, offering a top-down view of traffic. To our knowledge, there is no large public dataset from an infrastructure LiDAR perspective, which motivates using pre-trained models and transfer learning for such scenarios.

Deploying deep models on edge devices has gained attention as Edge AI becomes more capable. The NVIDIA Jetson AGX Orin, for example, provides GPU acceleration in a compact form factor and supports configurable power modes (15 W, 30 W, 50 W, etc.) to trade off performance and energy usage. Prior works in edge deployment often consider model compression (quantization, pruning) or efficient model architectures. Our approach is orthogonal and complementary: rather than modifying the network, we reduce the input data per frame via intelligent pre-processing. This can work alongside model-level optimizations. In the context of V2I, an edge device like a Jetson Orin could be installed at an intersection running a detection model; our method helps ensure it can run at sufficient frame rates even in low-power modes by skipping unnecessary computations on frames where the scene hasn't changed much.

3 Methodology

The goal of this pipeline is to maintain high 3D detection accuracy while reducing computational requirements per frame for real-time LiDAR data processing. The core idea is to avoid re-processing regions of the point cloud that likely contain no new information. This is achieved by an adaptive cropping mechanism guided by previous detections, as shown in Fig. 1.

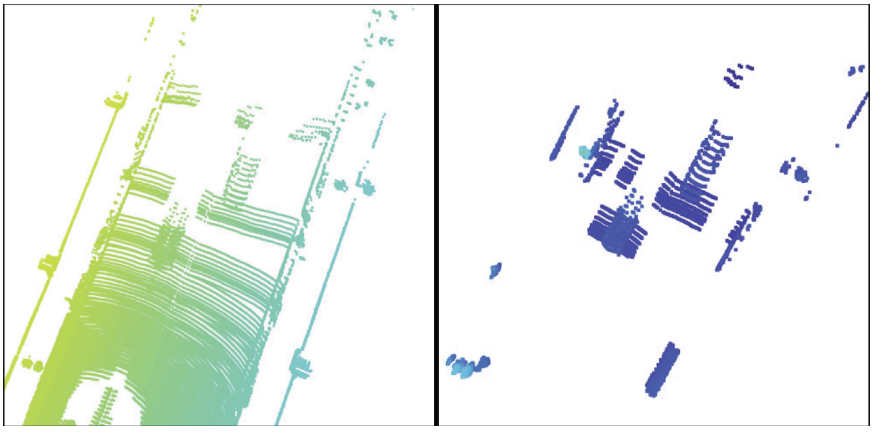

Fig. 1. Adaptive cropping concept. **Left:** Full LiDAR point cloud with detected objects (cars and pedestrians) at time t. **Right:** The point cloud at time $t + 1$, cropped to regions around the projected positions of previously detected objects.

3.1 Adaptive Cropping Strategy

Given a LiDAR point cloud at time t (frame t) and the set of object bounding boxes detected in that frame, we aim to focus the processing of the next frame $t+1$ only on those areas. Formally, let $B_t = \{b_{t,1}, b_{t,2}, \ldots, b_{t,N}\}$ be the 3D bounding boxes predicted in frame t. Each box $b_{t,i}$ provides the region where an object (e.g., a car or pedestrian) was detected, along with an estimated velocity $\mathbf{v}_{t,i} = (v_x, v_y)$ for the object, estimated lavaraging the vehicle state or directly by the network. For frame $t+1$, we predict an approximate new position for each object by a simple linear motion model:

$$\mathbf{c}_{t+1,i} = \mathbf{c}_{t,i} + \mathbf{v}_{t,i}\, \Delta t, \tag{1}$$

where $\mathbf{c}_{t,i}$ is the center of box on the 2D plane $b_{t,i}$ and Δt is the time between frames. This gives a translated bounding box $\tilde{b}_{t+1,i}$ for each prior detection, representing where we expect the object to be in the new frame. If the sensor is also moving, as in nuScenes, where the LiDAR is vehicle-mounted, we first perform *ego-motion compensation*. In particular, we use the vehicle's odometry to transform the previous detections into the coordinate frame of the new sensor position before applying the velocity translation. This ensures that the predicted positions account for the sensor's own movement.

Once we have the predicted boxes $\tilde{B}_{t+1} = \{\tilde{b}_{t+1,i}\}$, we crop the new frame's point cloud P_{t+1} by filtering out all points outside these regions. The result is a reduced point cloud P'_{t+1} that contains points only where we expect existing objects to be. This significantly decreases the number of points the detector needs to process. We denote this cropping function as:

$$P'_{t+1} = \mathcal{C}(P_{t+1}, \tilde{B}_{t+1}),$$

where $\mathcal{C}$ performs a geometric point-in-box test for each point against all boxes in $\tilde{B}_{t+1}$.

Because the entire operation can be heavily parallelized among all points, it introduces minimal overhead. We leverage tensor operations in PyTorch to check point inclusion in all boxes in parallel, rather than iterating per point or per box. This approach takes advantage of the GPU's capacity to handle many points simultaneously, offsetting the cost of the cropping step with a minimal runtime compared to the detector itself.

3.2 Expansion and Update Frequency

Two parameters govern the trade-off in the adaptive cropping pipeline: the *expansion factor* and the *update frequency*. The expansion factor $e \geq 1$ slightly enlarges each predicted box $\tilde{b}_{t+1,i}$ before cropping. This is to account for prediction errors or unexpected motion; e.g., if an object moves faster than predicted or the previous bounding box was slightly too tight. An expanded region ensures the object's points are still captured in P'_{t+1}. In particular, we scale the width, length, and height of each box by e (around its center). An acceptable value has

been experimentally computed as between $e = 1.5$ and 2.0, determined empirically as a balance between including enough points to compensate for detection errors and not including too many extra points that would make the preprocessing phase ineffective.

The update frequency refers to how often we allow a full point cloud to be processed without cropping. If we crop every single frame based only on the previous frame's detections, a new object that appears (e.g., a car turning onto the road from a side street) would not be detected because its points would be cropped out–there was no prior box for it. To handle such cases, we periodically run an uncropped inference. For instance, a strategy of "no cropping every five frames" means that out of each five consecutive frames, one frame (every fifth) is processed normally by the detector on the whole point cloud, and the other four frames are processed with cropping. This periodic full frame acts as a refresh, catching any new objects or correcting drift if an object's predicted box was off. A higher frequency of full processing (e.g., every 2 or 3 frames) yields higher accuracy but less speed-up, whereas a lower frequency (every 10 or 20 frames) maximizes efficiency at the risk of accuracy loss. The trade-off between higher and lower update frequencies is evaluated in Sect. 4.

3.3 System Integration

We implemented the proposed method in a modular pipeline. The LiDAR sensor data is first collected through the Robot Operating System (ROS) for our custom dataset, which publishes a point cloud message at 10 Hz. We convert the ROS point cloud message to a NumPy array of point coordinates using a custom optimized routine (bypassing per-point Python loops by directly mapping the binary data in memory). The detector model (e.g., SECOND or PV-RCNN) is then applied either on the full cloud or the cropped cloud depending on the frame index and chosen strategy. The model's predicted bounding boxes and velocities are then used for the next frame's cropping as described above. This cycle repeats in real-time. The overall architecture thus consists of a Detector node and a Cropping module that work in tandem: the Detector provides predictions to the Cropping module, which then pre-processes the next frame for the Detector.

By adjusting the power mode of the NVIDIA Jetson AGX Orin, we can simulate different edge deployment scenarios from energy-saving (15 W) to maximum performance (50 W). Our software leverages CUDA acceleration for both the detection model and the cropping operation. Notably, because the detection models we use (Sect. 4) are able to output object velocities as part of their detection head, we do not perform an explicit data association or tracking step between frames–the model's output directly feeds the cropping logic. This simplifies the system compared to a full tracking-by-detection pipeline. In cases where a detector might not provide velocity, one could integrate a light-weight tracking algorithm to estimate velocities of detected objects.

4 Experiments and Results

Experiments are performed using two datasets: the first one is a custom infrastructure LiDAR dataset we collected for an edge deployment scenario, while the second is the public nuScenes dataset for a more diverse and large-scale evaluation. In the test, we use different detector models to show that our cropping strategy is model-agnostic.

4.1 Evaluation on NuScenes Dataset

NuScenes [1] is a challenging dataset collected with a 32-beam LiDAR on a moving vehicle in urban environments (captured in Boston and Singapore). It provides annotated 3D bounding boxes for 23 classes, though for this study we focus on common classes *Car* and *Pedestrian* for which we have pre-trained detectors. We use the nuScenes *mini* split (a smaller subset with 10 scenes) for evaluation due to ease of experimentation and because our interest is in comparative performance (with vs. without cropping) rather than maximizing absolute accuracy.

We evaluated two detection models on nuScenes:

- **SECOND-MultiHead** [7]**:** a one-stage voxel-based detector (with a multi-head detection layer specialized for different classes). We use a variant configured for nuScenes data. It produces class-specific detections along with estimated velocities for each object. This model has about 35 million parameters.
- **VoxelNeXt** [2]**:** a recent fully sparse convolutional detector, also one-stage, with roughly 31 million parameters, known for its efficiency and high accuracy on nuScenes. It also predicts object velocities.

We apply our adaptive cropping strategy with varying settings: the cropping update frequency is set to every 2, 3, 5, 10, or 20 frames (meaning a full point cloud is processed at those intervals), and expansion factor e is set to 2.0 by default, with some tests at 1.5 or 3.0 for extreme cases. We measure the mean Average Precision (mAP) for Car and Pedestrian classes under each setting, and compare to the baseline mAP when every frame is processed fully (no cropping). We also compute the fraction of points processed relative to the baseline (as an indicator of computational load).

Table 1 summarizes the results for the SECOND-MultiHead model. As the cropping becomes more aggressive (full processing less frequent), the mAP drops gradually, but the fraction of points processed decreases significantly:

From Table 1, we see that processing one out of every 2 or 3 frames fully (i.e., cropping on intermediate frames) yields a very minor mAP loss (under 2–4% points) while cutting the point cloud workload to around 50–60%. More aggressive settings like cropping 9 out of 10 frames (only one full frame per second, since nuScenes LiDAR is 20 Hz) start to incur noticeable accuracy drops, especially for the Car class (which moves faster and is more affected by staleness

Table 1. Detection performance of SECOND-MultiHead on nuScenes (Car and Pedestrian) under different cropping strategies. mAP is shown for Car/Ped; ΔmAP is the drop from baseline, and %Points is the fraction of points processed relative to no cropping. Expansion factor is 2 unless noted.

Cropping Strategy	mAP (Car/Ped)	ΔmAP	%Points
Baseline (no cropping)	0.867 / 0.867	0.00 / 0.00	100%
Crop every 2 frames	0.846 / 0.860	−0.021 / −0.007	61.53%
Crop every 3 frames	0.829 / 0.853	−0.038 / −0.014	50.34%
Crop every 5 frames ($e = 1.5$)	0.770 / 0.814	−0.097 / −0.053	33.42%
Crop every 5 frames ($e = 2$)	0.796 / 0.852	−0.071 / −0.015	39.84%
Crop every 10 frames ($e = 1.5$)	0.677 / 0.718	−0.190 / −0.149	24.66%
Crop every 10 frames ($e = 2$)	0.715 / 0.811	−0.152 / −0.056	32.04%
Crop every 20 frames ($e = 2$)	0.548 / 0.762	−0.319 / −0.105	25.99%
Crop every 20 frames ($e = 3$)	0.710 / 0.838	−0.157 / −0.029	43.40%

in detection). Using a larger expansion factor can recover some accuracy in those extreme cases, as seen with the 20-frame case: increasing e from 2 to 3 improved Car mAP from 0.548 to 0.710, at the cost of processing more points (43% vs 26%).

We observed similar trends with VoxelNeXt. Its overall baseline accuracy is higher (baseline mAP 0.886 Car, 0.909 Ped), but under cropping it behaved analogously: moderate cropping leads to marginal drops. Due to space, we omit the full table, but highlight a few points: cropping every three frames with VoxelNeXt yielded mAP $\approx$ 0.855 (Car) / 0.892 (Ped) with about 50% points, and cropping every 5 frames (with $e = 2$) still gave 0.826 / 0.892 at 41% points. These results demonstrate that our method generalizes across detectors. In summary, leveraging temporal redundancy can substantially reduce computation while preserving detection performance on nuScenes. Figure 2 visualizes the trade-off between mAP loss and fraction of points processed for different strategies, showing that the curve remains flat until very high levels of cropping with a significant reduction of processed points.

4.2 Edge Deployment on Custom Dataset

Our custom dataset was collected using an Ouster OSDome-128 LiDAR mounted at about 6 m height overlooking a road. It captures a short-range, top-down view of vehicles and pedestrians. The sensor operates at 10 Hz. Because this dataset is small and lacks exhaustive annotations, we rely on models pre-trained on KITTI for detection, which cover Car and Pedestrian classes. In this scenario, we tested two detection models, which present significant differences in terms of architecture. SECOND is a single-stage architecture, while PV-RCNN is a two-stage model. We ran these on a Jetson AGX Orin Developer Kit, testing in three power modes: 15 W, 30 W, and 50 W (max). We compare the baseline

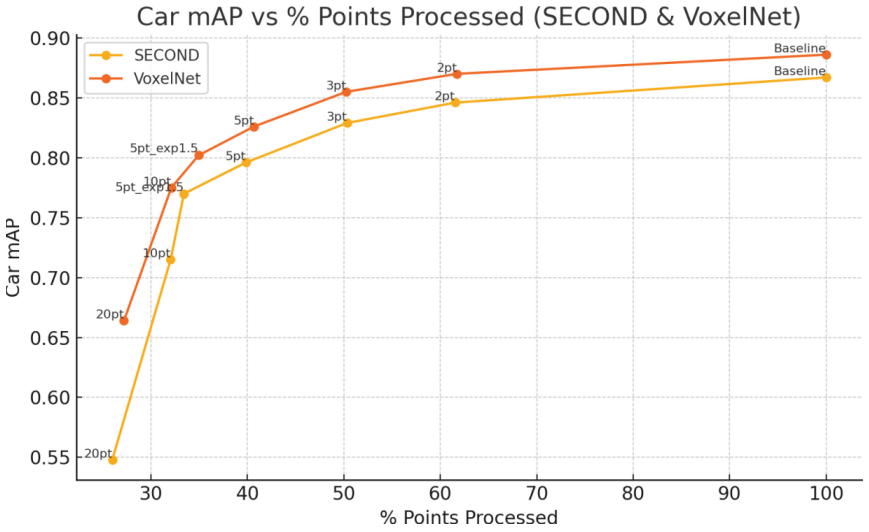

Fig. 2. Trade-off between accuracy and data reduction on nuScenes. Plot shows mAP vs. fraction of points processed for various cropping frequencies (numbers next to points indicate frames skipped). Both SECOND and VoxelNeXt exhibit graceful degradation, with moderate cropping (skipping 1-3 frames) giving large data reduction for small mAP loss, whereas very infrequent full updates (skipping more than frames) lead to steeper accuracy drop.

inference speed (processing every frame fully) against our cropping approach of processing one full frame every 10 frames (i.e., 1 Hz full, 9 out of 10 frames cropped). This setting was chosen based on the results of the NuScene dataset, and on the relatively low motion of objects in the scene (e.g., vehicles stopping at a crosswalk, parking, etc.), and to maximize efficiency on the edge device.

Table 2 shows the achieved inference rates in frames per second (FPS) for each model and power mode, with and without adaptive cropping. We report the effective throughput, including the cropping overhead.

Table 2. Inference speed (Hz) on NVIDIA Jetson AGX Orin for SECOND and PV-RCNN models, with and without adaptive cropping (cropping 9/10 frames). Higher is better.

Power Mode	SECOND (1-stage)		PV-RCNN (2-stage)	
	Base	+Cropping	Base	+Cropping
15 W (Low)	4.87 Hz	5.93 Hz	1.26 Hz	2.09 Hz
30 W (Med)	7.98 Hz	8.18 Hz	1.85 Hz	3.08 Hz
50 W (Max)	9.37 Hz	8.52 Hz	2.43 Hz	4.27 Hz

As expected, the heavier two-stage model PV-RCNN benefits substantially from the cropping strategy. At the lowest power mode (15 W), PV-RCNN can only process approximately 1.3 frames per second without cropping, which falls below real-time requirements. With cropping, its speed increases to over 2 Hz, a 66% improvement. At 30 W, PV-RCNN goes from 1.85 to 3.08 Hz, and at 50 W from 2.43 to 4.27 Hz, a 76% gain. The one-stage SECOND model, being much faster, sees less dramatic changes. Interestingly, at 50 W, SECOND runs slightly slower with cropping (8.52 Hz vs 9.37 Hz). This happens because, at maximum performance, the model processes the full 64-beam cloud quickly enough that the cropping overhead and the smaller batches of points cause less GPU utilization. In lower power modes, however, SECOND does see improvements: at 15 W, 4.87 to 5.93 Hz (22% gain). This indicates that our method is most useful when the detector is bottlenecked by computation; for a very lightweight model or a scenario with few points, cropping might not help and can even hurt slightly due to overhead.

Qualitatively, we verified that in our custom scene, cropping every 10 frames did not miss any vehicles or pedestrians–the scene is relatively static or slow-moving, and the model's predictions remained stable. Our approach effectively allowed the Jetson device to analyze the scene at roughly the sensor frame rate (10 Hz) using the simpler model, or at a playable frame rate using the complex model, all while operating in low-power configurations that would be suitable for an autonomous infrastructure sensor.

4.3 Results Analysis

The experiments on nuScenes and the custom setup show a consistent pattern: exploiting temporal redundancy via adaptive cropping can yield significant computational savings with minimal impact on detection performance, up to a point. The nuScenes results highlight that modest cropping frequencies (up to 3 frames) are quite safe, incurring very small accuracy penalties. This is promising for real deployment, as it suggests one could double the frame rate or halve the hardware requirements and still meet accuracy targets. The trade-off becomes more pronounced with aggressive cropping, skipping 10–20 frames, where one must also increase the expansion factor to avoid missing fast objects, thereby clawing back some of the computational savings. Indeed, the optimal strategy can be tuned based on the scenario: fast highway traffic might use a lower skip (e.g., every 2–3 frames), whereas a slow environment (parking lot or our RSU scenario) can tolerate a higher skip (every 5–10 frames).

On the edge device side, the cropping strategy is particularly beneficial for heavy models that generally would be too slow on the given hardware. It essentially acts as an external scheduler, dynamically reducing the model's workload. An interesting observation is that cropping provides diminishing (or no) returns for very efficient models at high power, indicating that the overhead of our approach is not zero.

5 Conclusion and Future Work

In this work, we presented an adaptive cropping strategy for real-time LiDAR object detection on edge devices. By leveraging temporal consistency in point cloud streams, our method reduces the data processed per frame and accelerates inference without requiring any changes to the core detection model. Through experiments on a custom RSU-mounted LiDAR dataset and the nuScenes benchmark, we demonstrated that this approach achieves significant efficiency gains, up to 50% fewer points, or 2× speed, with only marginal losses in detection accuracy for moderate settings. On an NVIDIA Jetson Orin edge platform, the technique enabled heavy 3D detectors to run at substantially higher frame rates, validating the feasibility of deploying efficient 3D perception pipelines on low-power hardware. This result is encouraging for future V2I applications, where roadside sensors could enhance vehicle perception in real-time.

Acknowledgments. This paper was supported by "Sustainable Mobility Center (Centro Nazionale per la Mobilità Sostenibile – CNMS)" project funded by the European Union NextGenerationEU program within the PNRR, Mission 4 Component 2 Investment 1.4.

References

1. Caesar, H., et al.: nuscenes: a multimodal dataset for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11621–11631 (2020)
2. Chen, Y., Liu, J., Zhang, X., Qi, X., Jia, J.: Voxelnext: fully sparse voxelnet for 3D object detection and tracking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 21674–21683 (2023)
3. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3354–3361. IEEE (2012)
4. Liu, X., Qi, C.R., Guibas, L.J.: Flownet3D: learning scene flow in 3D point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 529–537 (2019)
5. Shi, S., et al.: PV-RCNN: point-voxel feature set abstraction for 3D object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10529–10538 (2020)
6. Sun, P., et al.: Scalability in perception for autonomous driving: Waymo open dataset. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2446–2454 (2020)
7. Yan, Y., Mao, Y., Li, B.: Second: sparsely embedded convolutional detection. Sensors **18**(10), 3337 (2018)
8. Yin, T., Zhou, X., Krahenbuhl, P.: Center-based 3D object detection and tracking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11784–11793 (2021)