IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

SpikeClouds: Streaming Spike-based Processing of LiDAR for Fast and Efficient Object Detection

Michael Neumeier[1,4], Nael Fasfous[2], Bing Li[3] and Axel von Arnim[1]

***Abstract*—LiDAR sensors are used to provide three-dimensional information about the environment in many robotics applications. The information, accumulated in 3D point clouds, is first acquired by the sensor and then processed further, which leads to high end-to-end latencies and large memory footprints. Streaming approaches tackle this problem by processing partial point cloud data during scanning of the environment. In contrast to existing work that is limited to power hungry, rotating mechanical scanners, in this paper, we present a streaming method for more efficient scanline-based LiDAR sensors. We process the sequence of scanlines in form of SpikeClouds with a Spiking Neural Network (SNN) backbone and perform 3D object detection from the accumulated information using a Convolutional Neural Network (CNN) head. Our method achieves close to state-of-the-art detection performance on datasets KITTI and JRDB22 while reducing the end-to-end latency by 10% and the average memory footprint by 95% on standard GPU hardware. Additionally, when ported onto neuromorphic hardware, our backbone requires 25× less energy compared to reference backbones. SpikeClouds achieves fast and efficient environmental perception for robotic applications by streaming LiDAR to enable spike-based processing.**

***Index Terms*—Range Sensing, Object Detection, Neurorobotics, LiDAR Processing, Spiking Neural Networks.**

I. Introduction

THREE dimensional data from LiDAR (Light Detection And Ranging) sensors are a rich source of environmental information for applications in autonomous driving and robotics. If processed efficiently, the information produced by a high-resolution LiDAR can enrich the semantic understanding of a scenario far beyond what is possible using only standard RGB cameras [1]. Furthermore, information about distance to unknown objects can be accurately obtained from the sensor's data without semantic understanding, adding more robustness to safety critical tasks. When combined with standard cameras, sensor fusion techniques further improve the accuracy and precision of computer vision algorithms [1]. Considering the rapid progression in development of more robust LiDAR sensors at better resolutions, lower costs and power consumption, it is likely that they become a standard sensor for many complex robotic perception applications.

Manuscript received: February, 28, 2025; Accepted: April, 29, 2025.

This paper was recommended for publication by Editor Tetsuya Ogata upon evaluation of the Associate Editor and Reviewers' comments.

[1]First Author and Forth Author are with fortiss GmbH, Neuromorphic Computing, Munich, Germany neumeier@fortiss.org

[2]Second Author is with BMW Group, Munich, Germany

[3]Third Author is with University of Siegen, Digital Integrated Systems Group, Siegen, Germany

[4]First Author is with Technical University of Munich, TUM School of CIT, Munich, Germany

Digital Object Identifier (DOI): see top of this page.

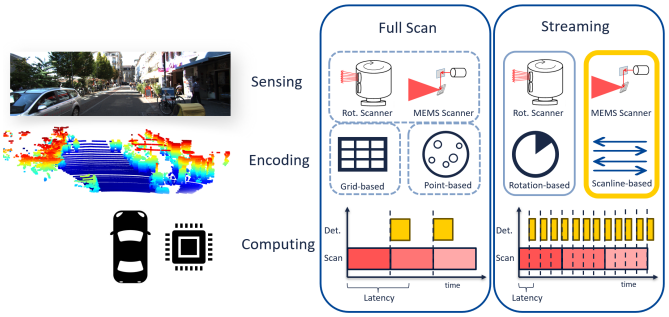

Fig. 1. **LiDAR full scan vs. streaming processing.** Our introduced scanline-based processing of MEMS-based LiDAR technique is highlighted in yellow. Scan, detection and total latencies are qualitatively evaluated in processing diagrams.

LiDAR is a technology to measure distances with pulsed light emitted by a laser. A physical scanning system operates on top of the rangefinder principle and steers the laser beam at different azimuths and elevation angles. Motorized rotating scanners produce 360° horizontal field-of-view (FoV) by rotation and vertical FoV by stacking rangefinder units or nodding mirrors [2]. MEMS (Micro-Electro-Mechanical Systems) scanners steer the laser beam with electrically-controlled, integrated horizontal and vertical mirrors. This tight integration reduces both size and power consumption of the sensor [3]. The perceived information per scan is represented by a three-dimensional set of points, also called point clouds.

Most LiDAR-based approaches process the *full scan* 3D point cloud data (left column in Figure 1). Voxel-based methods divide the point cloud in a three dimensional grid which can be processed with 3D CNNs [4]. These approaches leverage regular memory accesses, but have a high memory footprint and computational cost [5]. Other approaches project the 3D data onto a 2D grid in order to make use of more efficient 2D CNNs [1], [6]. Point-based approaches apply computations only to the sparse set of points [7]. This leads to a smaller memory footprint, but introduces the challenges of irregular memory access patterns and dynamic kernel generation [5]. For all full scan methods the total time for an algorithmic result is the sum of scanning and processing time which leads to high end-to-end latencies [8]. Beyond that, accumulating full scans into data packets also heavily contributes to the in-car bandwidth requirements [9].

In contrast, *streaming* approaches exploit the fact that LiDAR scanners acquire environmental information over a certain period of time. They already start processing 3D point cloud slices during the accumulation of a full scan (right

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

column in Figure 1). Hence, they reduce the overall detection latency and peak bandwidth [8], [10], [11]. But the proposed streaming methods are all tailored to rotating, stacked LiDAR scanners. They are not compatible with the smaller and more efficient sensors like scanline-based MEMS LiDARs [3].

Spiking neural networks (SNNs) process sparse stream input events and trigger sparse sequence of output events based on internal neuronal dynamics, such as leaky integrate-and-fire (LIF) [12]. SNNs have been proven to provide efficiency gains, if the input data is sparse and spatio-temporal in nature, like streams from event-based vision sensors [13]. When SNNs are implemented on neuromorphic accelerators, which can support sparse binary activations and stateful neurons, improvements in energy-efficiency and latency can be gained over CPU/GPU implementations [14].

In this paper, we propose a scanline-based streaming method for LiDAR sensors using SNNs. We develop the SpikeClouds encoding technique to enable efficient, small-scale SNN backbones to preprocess the streamed sensor data. This reduces latency and bandwidth requirements due to the streaming operating principle, also on standard GPU hardware. Further benefits can be extracted when neuromorphic hardware is available. The accumulated and compressed SpikeCloud features are further processed by a CNN neck and head to detect objects. We test the approach on two real-world datasets, KITTI [15] and JRDB22 [16].

We summarize the contributions of this work as follows:

1) SpikeClouds, a streaming, spike-based encoding technique for low latency LiDAR processing and up to 95% peak sensor-to-accelerator bandwidth reduction by near-sensor processing of scanline updates.
2) A hybrid SNN-CNN SpikeCloudNet for 3D object detection with up to 10% end-to-end latency reduction on GPU for KITTI and JRDB22 datasets by hiding backbone latency within the scan accumulation time.
3) Neuromorphic near-sensor acceleration for additional latency reduction while requiring 25× less energy compared to reference approaches on edge GPUs.

In Section II, an overview of related work is presented, followed by a deep dive into our SpikeCloud methodology in Section III. In Section IV the presented methods are evaluated by first conducting an ablation study on their parametrization followed by a comparison to the state of the art in terms of object detection performance, latency, bandwidth and energy consumption. Finally, we draw our conclusions in Section V.

II. Related Work

A. Efficient LiDAR Processing

One pathway to efficient processing of LiDAR is to project 3D point clouds to a 2D grid based on the front view [17] or the bird's-eye-view projection [1], [6]. For a front view projection, a 2D point map with axes corresponding to azimuth and elevation angle is constructed [17]. In contrast, the bird's-eye-view (BEV) projection keeps the physical coordinates of depth and width. Typically BEV features are calculated from occupancy or density, height and intensity information per grid element [1], [6]. These feature maps are used as inputs to 2D CNNs for object detection. PointPillars propose a trainable encoding using a point-based layer. They outperform the hand-crafted approaches in object detection accuracy [18].

Another pathway is to leverage the intrinsic sparsity of point clouds. Therefore, the 3D grids are not represented with dense tensors, but instead in form of sparse tensors containing only non-zero grid elements. Hence, also the convolutions are sparsely computed for these active elements [19].

Recent work has combined a PointPillar projection with sparse 2D convolutions and shows advancements in both accuracy and latency [20]. For 3D object detection they propose an efficient one-stage method which is based on predicting object center heatmaps and regressing bounding box parameters [20], [21]. Nevertheless, all of these approaches have to wait for the accumulation of the complete point cloud until processing can be started. We use the PillarNet object detector [20] as a full scan reference. Our approach is based on their efficient object detector, but uses spiking instead of sparse convolutional layers to additionally support temporal processing.

B. LiDAR Streaming Processing

Instead of processing complete 3D point clouds, streaming-based methods are updated sequentially for a subset of perceived points [8]. Point cloud slices are also projected to the bird's-eye-view and processed by 2D CNNs. By adding LSTM layers and stateful prediction, objects that overlap several sectors can be accurately detected [8]. STROBE [11] additionally uses features from the previous scan by maintaining multi-scale memory feature maps. PolarStream [10] keeps the slices' natural representation in polar coordinates and introduces the polar pillar projection. These streaming-based approaches significantly reduce the latency of object detection while also reducing the peak computational load [8]. By sequentially consuming angular slices of points, they particularly target rotating scanners with stacked rangefinder units. For other LiDAR scanners, e.g. MEMS mirror-based ones, they are not directly applicable as the slice increments are different. Our work closes this gap by proving a method for scanline-based streaming for emerging LiDAR scanners.

C. SNNs for LiDAR

A small number of previous approaches used SNNs for LiDAR processing. They also project point clouds to the bird's-eye-view grid and encode them into spike trains based on temporal [22] or rate [23] coding. However, these techniques rely on processing a completed full-scan instead of stream-based processing. Both take inspiration from one-stage approaches to design SNN object detectors and report accuracy on the KITTI benchmark [15]. The idea of streaming real-time object recognition from raw LiDAR signals with SNNs is introduced in [24]. The sequence of perceived points is encoded to spikes, based on temporal, radial basis function-based or rate coding. A compact recurrent SNN performs object classification starting already *within* the scanning process [24]. In our work, we reformulate this idea for scanline-based LiDAR sensors, extend it towards 3D object detection tasks and provide hardware benchmarks.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

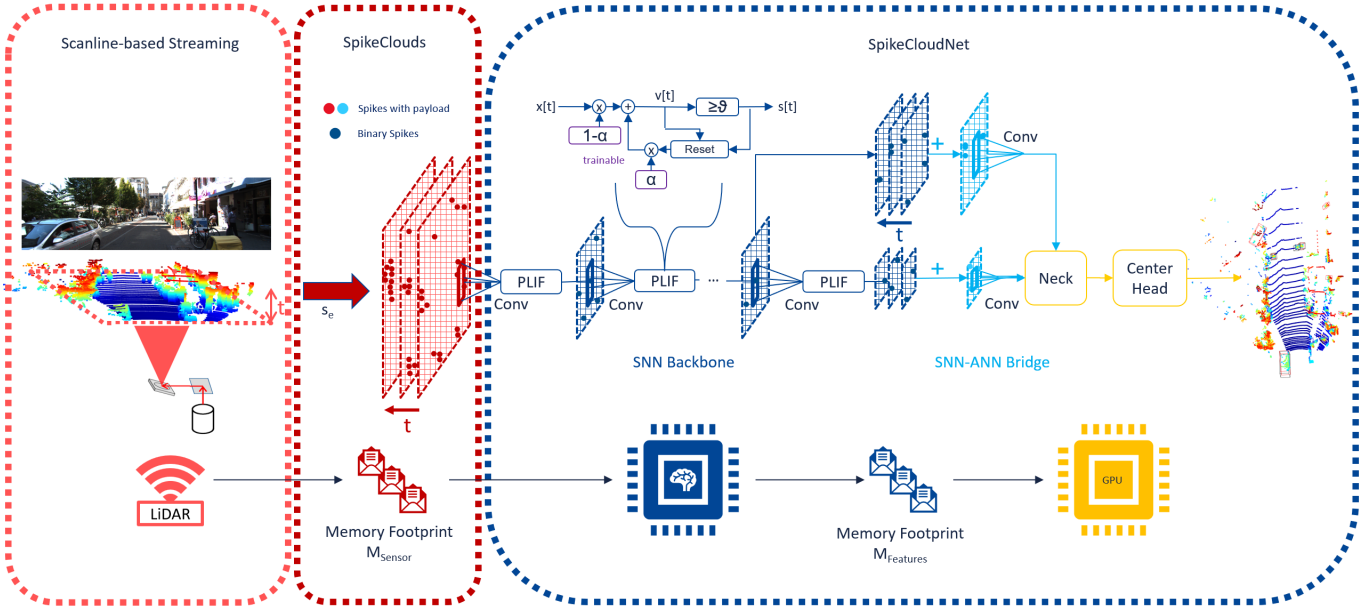

Fig. 2. **SpikeClouds System Overview**. From left to right: Scanline-based streaming of LiDAR sensor data, SpikeClouds bird's-eye-view projection and encoding and SpikeCloudNet as hybrid SNN-ANN for 3D object detection. Memory footprints M_{Sensor} and $M_{Features}$ as well as processing hardware for the distributed setup are shown below.

III. Methodology

A. System Overview

In Figure 2 we visualize our main modules. Scanline-based LiDAR streaming (III-B) enables the streaming processing of non-rotating LiDAR scanners. We generate SpikeClouds (III-C) by projecting the scanlines to the bird's-eye-view and encode active grid elements into spikes. This keeps the memory footprint of scanline packets small which reduces the sensor-to-processor bandwidth requirements. SpikeCloudNet (III-D) sequentially processes the SpikeClouds and detects objects in 3D. The SNN backbone is updated on every scanline iteration. If this update can be computed fast enough, the backbone processing latency can be hidden within the scanning time. We emphasize the combination of a streaming SNN backbone and a CNN detection neck and head. The latter are executed once per scan. In order to train this object detector, we introduce a hybrid training framework (III-E). For inference, next to an end-to-end GPU implementation (III-F1), we also set up a near-sensor neuromorphic processor setup (III-F2) which is also visualized in Figure 2.

B. Scanline-based LiDAR streaming

We propose a streaming processing method with focus on scanners with fast horizontal scanning (0.5-2 kHz) and slow vertical scanning (10-30 Hz) like modern MEMS LiDARs [3]. By iteratively accumulating information from horizontal scanlines, complete FoV point clouds are captured. Instead of waiting for the accumulation of the complete point cloud, we process each scanline in a streaming manner. We denote a complete point cloud P as a temporal sequence of scanlines S_t: $P = \{S_1, S_2, ..., S_T\} = \{S_t\}_{t=1}^{T}$, where Δt is the waiting time for a new scanline update (determined by the horizontal scan rate) and T is the total scan time (controlled by the vertical scan rate). Each scanline update returns points with a new elevation, which we translate to a slice in the vertical axis z. We end up with a simulated scanline sequence $\{S_z\}_{z=1}^{Z}$ with the granularity of each scanline being Δz and the overall vertical extent Z. This scanline sequence is obtained for the vertical mirror operated in non-resonant mode, in which the laser beam is steered from bottom to top. For the resonant mode, z elements are permuted according to the oscillation pattern of the vertical mirror, starting from medium height and gradually oscillating towards the minimum/maximum height and back.

C. SpikeClouds

Each scanline S_t contains points p_i with coordinates x_i, y_i, z_i and reflectivity r_i. We rasterize them to a bird's-eye-view grid B_j with grid elements j whose resolution is determined by the grid element size Δx and Δy. Using a mapping function $B_j = f_{SC}(p_i, \Delta x, \Delta y)$, a set describing all scanline points mapped to a specific grid cell element.

$$S_{i \to j} = \{p_i | B_j = f_{SC}(p_i, \Delta x, \Delta y)\} \tag{1}$$

The payload value of each active grid element is calculated with the encoding function s_e. We propose multiple encoding functions based on occupancy $s_{occ.}$, density $s_{dens.}$ and additional average height and reflectivity $s_{comb.}$ per grid element. For the combined encoding we stack the separate grids in the channel dimension, which leads to a bird's-eye-view grid information per scanline $S_t = \{s_e(B_j)\}$. Hence, the temporal sequence of projected scanlines is $SC = \{S_t\}_{t=1}^{T}$, which we call a SpikeCloud. We do not convert each scanline to a dense grid tensor, but keep up the sparse representation of active grid elements. This keeps the memory footprint per scanline low and directly supports an implementation on neuromorphic

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

hardware. Therefore, the sequence of scanline information can also be viewed from the perspective of the address event representation (AER) [25].

D. Hybrid SpikeCloudNet

We propose a hybrid SNN-CNN-based architecture (see Figure 2) to process the temporal sequence of scanlines. A spiking convolutional backbone is operated in a streaming manner and updated upon every new scanline. The SNN backbone consists of N blocks, each of them constructed by 2D convolution (3x3 kernel size), batch normalization and Parametric Leaky-integrate-and-Fire (PLIF) neurons. Every second 2D convolution operation has a stride of 2 in order to spatially compress the BEV-projected LiDAR data. PLIF neurons introduce a per layer trainable decay factor α_v which supersedes the manual decay factor tuning and enhances accuracy [26]. The neuron's state is reset to zero when it emits a spike (hard reset). Further, we also apply the decay factor on the neuron's input $x[t]$. Following [27], we divide the neuron computation in three stages: Neuronal charge (Eq.2) to integrate and decay the membrane potential $v[t]$, neuronal firing (Eq.3), in which the potential is compared to a constant threshold ϑ and neuronal reset (Eq.4) of the membrane potential, if the threshold is exceeded.

$$v[t] = \alpha_v v[t-1] + (1-\alpha_v)x[t] \tag{2}$$

$$s[t] = v[t] \geq \vartheta \tag{3}$$

$$v[t] = v[t](1-s[t]) \tag{4}$$

The recurrent charging of the membrane potential $v[t]$ allows for temporal information processing, while firing and resetting ensure sparse, binary information $s[t]$ to be communicated between consecutive layers n and $n+1$. We propose two SNN backbone variants, N=5 and N=7. Each variant outputs spikes at two different spatial compression ratios (for N=5: $\{4, 8\}$ and N=7: $\{8, 16\}$). The SNN-ANN bridge produces two real-value SpikeCloud feature maps at these ratios. Therefore, it sums up the binary spikes over the scanline sequence and applies 2D convolutions spike sums. These feature maps can be further processed with the multi-scale neck and center-based object detection head as proposed in [20]. The hybrid approach enables efficient, recurrent accumulation of scanline data (SNN backbone) while delivering accurate object detections (CNN neck and head).

E. Training Framework

We base our training framework on top of the standard point cloud training framework Det3D [28]. This enables direct comparability to point cloud based reference approaches on available benchmark datasets. It also makes use of advanced augmentation strategies which help boosting the detection accuracy. Therefore, we add the scanline-based SpikeClouds as a point cloud reader by simulating scanline sequences $\{S_t\}_{t=1}^{T} = \{S_z\}_{z=1}^{Z}$. In order to train the SNN backbone part, we integrate backpropagation-through-time (BPTT) and surrogate gradients [27], [29]. This makes an end-to-end gradient-based training of our hybrid SpikeCloudNet possible. For training, we scatter the sequence of sparse scanline grid elements to a dense 4-dimensional tensor with x, y, e, t dimensions. The SNN backbone is represented in multi-step mode [27] in order to parallelize and speed up the training. The memory footprint of the SpikeCloud features and both latency and energy efficiency on neuromorphic hardware heavily depend on the sparsity of spiking activity [30]. We use the AdamW optimizer with weight decay for weight sparsity. Furthermore, we add a per spiking layer regularization term to the overall loss function L_{total} from [20] for activation sparsity. Hence, with our loss function L_{SC} we jointly optimize object classification (cls), bounding boxes (iou, $bbox$) and activation sparsity (reg) with λ as a per loss weighting factor:

$$L_{SC} = \lambda_{cls}L_{cls} + \lambda_{iou}L_{iou} + \lambda_{bbox}L_{bbox} + \lambda_{reg}L_{reg} \tag{5}$$

F. Inference Setup

For real-time streaming processing the backbone update time has to be smaller than the scanline update time Δt. Hence, we provide inference recipes for two different hardware setups to speed up the SNN backbone updates.

1) End-to-end GPU: The first one is for fast inference on GPU. This does not require access to specific neuromorphic hardware, which enables usage for all kinds of robotic platforms. The GPU inference model runs in a streaming fashion by sequentially processing scanline data during FoV scanning. In order to do so, we execute the SNN backbone in single step mode [27]. As the underlying GPU cannot leverage sparsity to speed up computation, we process the spiking activations in form of dense tensors and set $\lambda_{reg} = 0$. For optimized inference, we fuse the BatchNorm parameters into 2D Conv weights and biases, use JIT-scripted PLIF neuron modules and compile a single-step update into optimized kernels with torch.compile [31].

2) Neuromorphic near-sensor processor and GPU: Inference on digital neuromorphic hardware such as Intel's Loihi 2 makes use of sparsity in activity, dataflow architecture of processing elements and the hardware-friendliness of computing with binary spikes. In order to fully exploit these on Intel's research platform Loihi2 [32], our neuromorphic inference model has to integrate specific hardware constraints. The Loihi2 convolutions do not support biases, hence we fuse the BatchNorm parameters into 2D Conv weights and the neuron biases. The convolution weights are quantized to 8-Bit in multiples of 2 in order to support Loihi's shift multiply operations. We apply this quantization and finetune for another 10 epochs to revive object detection accuracy. The trained SNN layers are partitioned and mapped onto Loihi's neuromorphic cores using Lava [33].

IV. Experiments

We first empirically study the best parametrizations for our approach. Then we evaluate the configurations for two datasets KITTI [15] and JRDB22 [16] along multiple metrics (accuracy, latency, memory footprints, energy-efficiency).

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

A. How to best parameterize SpikeClouds?

We perform parametrization studies for the SpikeClouds encoding and the SpikeCloudNet on the KITTI bird's-eye-view benchmark with reduced range of $X = [0, 40.8m]$ and $Y = [-20.4m, 20.4m]$. We train all models for 100 epochs on the training set and evaluate on the validation set. The object detection results are provided as a mAP value at a specific IoU per class (Car: 0.7, Pedestrian/Cyclist: 0.5). We report the average mAPs per class across easy, medium and hard difficulties.

1) SpikeClouds: First, we evaluate the SpikeClouds encoding function s_e. Therefore, we fix the grid sizes $\Delta x, \Delta y$ to $[0.075m, 0.075m]$ and the scanline size Δz to $[0.2m]$. As an object detector, we use the proposed SpikeCloudNet with N=5, including the SNN backbone, and a two-scale neck and center head. Table I shows results for three encodings s_e: occupancy, density and a combined encoding with density, average height and reflectivity. For the car class there are only minor differences between the three encodings. Pedestrians are detected most accurately with the density encoding. For cyclists, the additional height and reflectivity information in the combined encoding has a positive impact. For each s_e, we also compare resonant and non-resonant scanline modes. For both, the SpikeCloudNet can accurately detect objects. On average, the non-resonant mode leads to slightly better detection performance.

TABLE I
EVALUATION OF SPIKECLOUD ENCODING s_e.

s_e	Res.	Car	Pedestrian	Cyclist
occ.	-	88.6	43.3	68.7
occ.	y	88.6	46.2	69.3
dens.	-	**88.9**	**48.9**	70.0
dens.	y	88.0	44.3	67.1
comb.	-	88.5	43.1	**70.9**
comb.	y	88.3	42.4	70.3

Next, we study the impact of the scanline size Δz and the bird's-eye-view grid sizes $\Delta x, \Delta y$ on the detection performance. We keep the same SpikeCloudNet detector as in the previous experiment (N=5) and choose $s_{density}$ in non-resonating mode. Table II shows the detection performance, separated per class, and evaluated for different bin sizes. For all classes, a finer bird's-eye-view grid size $\Delta x, y$ leads to more accurate results. The scanline size Δz encodes the resolution and length of the scanline sequence. Here the trend towards smaller bin sizes is not kept up. The medium scanline size $\Delta z = 0.20m$ performs the best.

Based on these experiments, we continue with a SpikeCloud paramerization of $s_{density}$, $\Delta x, \Delta y = [0.075m, 0.075m]$ and $\Delta z = 0.20m$.

2) SpikeCloudNet: We study the architecture of the SpikeCloudNet detector. For the SNN backbone, the relevant hyperparameters are the channel size C of the SpikeCloud features and the number of blocks N. A backbone variant is denoted by SCNet^N_C. We also investigate the neck choice. We test approaches without neck, with one feature-sized (f1) and two feature-sized (f2) necks and center head (c). In order to test our hypothesis regarding the hybrid approach, we also compare

TABLE II
EVALUATION OF SCANLINE AND BIRD'S-EYE-VIEW GRID SIZES.

$\Delta z \downarrow$ $\Delta x, y \rightarrow$		0.075m	0.10m	0.15m
Car	0.10m	88.5	88.2	85.9
	0.20m	**88.9**	88.0	85.9
	0.40m	88.1	87.4	85.1
Ped.	0.10m	45.5	45.0	41.9
	0.20m	**48.9**	45.0	43.3
	0.40m	46.2	42.8	43.8
Cycl.	0.10m	69.6	68.6	64.4
	0.20m	**70.0**	66.8	62.3
	0.40m	64.8	65.3	60.0

versions with a spiking neck and/or a spiking detection head. These results are shown in Table III. SCNet$^5_{128}$ stands out as the best backbone choice. The non-spiking multi-scale feature neck, combined with a non-spiking head, improves performance.

TABLE III
EVALUATION SPIKECLOUDNET ARCHITECTURE.

Backbone	s	Neck	s	Head	s	Car	Ped.	Cycl.
SCNet$^5_{128}$	√	-		c		81.4	45.3	59.0
SCNet$^5_{128}$	√	f1		c		88.4	48.0	67.0
SCNet$^5_{64}$	√	f2		c		88.0	48.3	62.6
SCNet$^5_{128}$	√	f2		c		**88.9**	**48.9**	**70.0**
SCNet$^5_{256}$	√	f2		c		88.7	48.2	67.4
SCNet$^7_{128}$	√	f2		c		86.0	44.8	64.4
SCNet$^5_{128}$	√	f2	√	c		80.3	46.0	60.8
SCNet$^5_{128}$	√	f2	√	c	√	79.9	43.7	63.2

In the final study, we evaluate the spike activity regularization and its impact on sparsity, effective synaptic operations (SynOps), average feature memory footprint and detection accuracy. In Table IV, we see that both sparsity and SynOps improve for increased λ_{reg}. For larger, easier objects like cars the accuracy almost stays constant. For smaller, harder ones, pedestrians and cyclist, the accuracy drops up to 5%. We conclude that the additional spikes for small λ_{reg} encode fine details for small object classes.

TABLE IV
SPARSITY, EFFECTIVE SYNAPTIC OPERATIONS, MEMORY FOOTPRINTS AND MAPS FOR DIFFERENT SPARSITY REGULARIZATION WEIGHTS.

λ_{reg}	Sparsity	SynOps	M$_{Feat}$	Car	Ped.	Cycl.
0	0.846	6.3e10	1.65MB	**88.9**	**48.9**	**70.0**
0.5	0.982	9.3e9	0.17MB	88.5	48.0	67.8
1.0	0.990	4.8e8	**0.06MB**	88.3	46.4	65.8
2.0	0.992	4.5e8	0.19MB	87.8	45.5	67.2
5.0	**0.997**	**2.2e8**	0.09MB	88.1	46.5	64.4

B. Object Detection Performance

We evaluate our proposed SpikeCloudNet on KITTI [15] and JRDB22 [16] benchmarks. Based on the conducted parameterization study, we choose SpikeCloud $s_{density}$ encoding in non-resonant mode with a grid size $\Delta x, \Delta y = [0.075m, 0.075m]$ and scanline size $\Delta z = 0.20m$. As labels for the testing splits are not publicly available, we report

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

accuracies on the validation splits. Table V shows the object detection performances and total latencies for the two datasets.

1) KITTI: We train our SpikeCloudNet on 3712 training samples for 100 epochs and evaluate on 3769 validation samples. KITTI object categories are car, pedestrian and cyclist. We report object detection accuracies for both bird's-eye-view and 3D evaluation in mAPs at IoU 0.7 for cars and 0.5 for pedestrians and cyclists. The detection range is [0.0m, 70.0m] in X, [-40.0m, 40.0m] in Y and [-3.0m, 1.0m] in Z direction. In Table V, we compare our approach to the state-of-the-art. For both KITTI benchmarks (BEV and 3D) SpikeCloudNet performs better than other SNN approaches like Deep SCNN [22] and SpikiLi [23]. For most cases, it also outperforms legacy point cloud object detectors like Second [19]. When comparing to the state of the art PV-RCNN [34] and PillarNet [20], SpikeCloudNet stays within a boundary of 5% accuracy loss, but provides end-to-end latency improvements.

2) JRDB22: In order to show the generalization of our approach across application domains, we also evaluate SpikeCloudNet for a dataset collected with a social mobile manipulator, the JackRabbot platform [16]. We train for 50 epochs on the samples from the 22 training sequences and evaluate on samples from the 5 validation sequences. This dataset is human-centric, hence only ground truths for the pedestrian category are provided. The range is [-40.0m, 40.0m] in X, [-40.0m, 40.0m] in Y and [-2.0m, 2.0m] in Z direction. The mAP values are reported at three IoUs from 0.3 to 0.7. In Table V we evaluate these against the state-of-the-art. SpikeCloudNet with density encoding performs accurate 3D object detection at a similar level as the common point cloud approach PointPillars [18] across all IoUs. Adding additional height and reflectivity information to the SpikeClouds with combined encoding enhances accuracy, particularly for higher IoUs.

C. Latency

We define the **end-to-end latency** as the duration between the start of FoV scanning and the detection of an object. For *full scan* processing approaches, this latency is the sum of point cloud accumulation time and processing time. KITTI and JRDB22 data are both recorded with a 10Hz Velodyne LiDAR sensor, hence we assume the point cloud accumulation time to be 100ms [37]. The processing time is calculated from the reported FPS value or measured on Nvidia RTX 2080 for the PillarNet and SpikeCloud variants. SpikeCloudNet consists of a streaming, spiking backbone and non-streaming neck and detection head. We report the $\mathrm{T}_{Tot.}$ as the sum of times for point cloud accumulation, the final backbone update and the downstream detection. Furthermore, we visualize the latencies in form of pipeline plots for PillarNet and our SpikeCloudNet in Figure 3. It is clearly visible that most of the backbone computation in blue is hidden within the scan time in red. In Table V, we report the measured latency on Nvidia RTX 2080, following literature which reports FPS values on similar grade GPUs. For KITTI, SpikeCloudNet saves between 30-60ms compared to the most accurate state-of-the-art approaches.

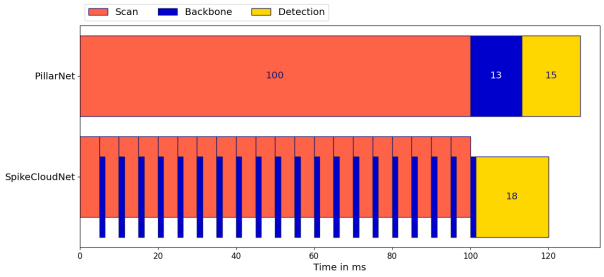

Fig. 3. **End-to-end latency pipeline plot** for PillarNet and SpikeCloudNet measured on Nvidia RTX 2080.

Compared to faster variants like PillarNet, it still reduces end-to-end latency by 8ms. Only SpikiLi outperforms our approach for the KITTI BEV benchmark. For JRDB22, the state-of-the-art approach RPEA [36] is already optimized for low latency. With our streaming processing and GPU inference optimizations, SpikeCloudNet can still outperform it by 5ms.

D. Sparsity, OPs and Memory Footprints

Spike-based processing also has an impact on compute operations and memory footprints (bandwidth requirements). In Table VI, we compare activation sparsity, multiply-accumulate (MAC) and accumulate (AC) operations for the backbones. The higher the regularization weight λ_{reg}, the higher the sparsity. We also showcase the hardware friendliness of SNNs here, as all convolutions with binary input activations can be performed as ACs. As a result the SCNet backbones need at least one order of magnitude less MACs than the PillarNet backbones. The regularization additionally helps reduce the number of operations. These results already indicate additional efficiency gains when using neuromorphic hardware to accelerate the backbone. We also report the average memory footprints of raw LiDAR data M_{Sensor} and backbone features M_{Feat}. Following [9] we calculate M_{Sensor} with 196 bits per point. Here we compare the complete point cloud memory footprint for reference approaches with the average memory footprint of scanlines for our streaming approach. This serves as a metric for the sensor to processor bandwidth requirements. Table VI highlights that by sending data in form of scanline packets their average memory footprint can be reduced from 0.45MB to 0.02MB. M_{Feat} consists of 16 bits for each x, y, channel index and for non-spiking features, an additional 32 bit payload. This results in 48 bits per feature spike and 80 bits for a non-spiking feature element. In the near-sensor computing setup, this metric evaluates the network traffic between backbone and detector accelerators. The high values for the non-streaming reference approaches only come into play when backbone and prediction modules are executed on separated processors. Table VI shows that the spike regularization also helps to keep M_{Feat} low.

E. Inference on edge platforms

Finally, we also benchmark our SpikeCloudNet in a constrained edge setup with reduced range, KITTI LiDAR input ($X = [0, 10.0m]$ and $Y = [-5.0m, 5.0m]$). We choose two

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

TABLE V
PERFORMANCE EVALUATION OF SPIKECLOUDNET ON KITTI AND JRDB22.

Dataset	Approach	Accuracy mAP			End-to-end Latency
		Car, @0.7 IoU	Ped., @0.5 IoU	Cycl., @0.5 IoU	$T_{Tot.}$ in ms
KITTI BEV	VoxelNet [4][2]	89.4, 79.3, 77.4	46.1, 40.7, 38.1	66.7, 54.8, 50.6	327.3
	Second [19][2]	88.1, 79.4, 78.0	55.1, 46.3, 44.8	73.7, 56.0, 48.8	150.0
	PV-RCNN [34][2]	**95.0**, **90.7**, 86.1	**59.9**, 50.6, 46.7	82.5, 68.9, 62.4	180.0
	PillarNet18S [20][1]	89.7, 88.6, **87.4**	55.2, **54.1**, **51.2**	**84.7**, **69.5**, **65.5**	155.8
	PillarNet18 [20][1]	87.6, 86.9, 86.6	44.0, 46.1, 44.3	81.1, 67.0, 63.0	128.2
	Deep SCNN [22]	86.5, 82.0, 79.1	50.3, 48.2, 46.5	75.6, 65.5, 63.2	128.0
	SpikiLi [23]	82.7, 75.5, 68.7	-	-	**103.0**
	SpikeCloudNet	89.8, 84.1, 82.7	50.3, 47.1, 43.6	78.4, 63.0, 59.8	120.3
KITTI 3D	VoxelNet [4][2]	77.5, 65.1, 57.7	39.5, 33.7, 31.5	61.2, 48.4, 44.4	327.3
	Second [19][2]	83.1, 73.7, 66.2	51.1, 42.6, 37.3	70.5, 53.9, 46.9	150.0
	PV-RCNN [34][2]	**90.3**, **81.4**, **76.8**	**52.2**, 43.3, 40.3	78.6, 63.7, 57.7	180.0
	PillarNet18S [20][1]	85.4, 78.5, 76.0	45.3, **45.0**, **41.3**	**81.3**, **64.8**, **61.2**	155.8
	PillarNet18 [20][1]	83.4, 77.1, 75.4	38.8, 39.6, 37.7	78.6, 63.7, 60.8	128.2
	Deep SCNN [22]	71.8, 67.4, 65.6	47.1, 42.5, 39.4	69.2, 59.2, 55.3	128.0
	SpikeCloudNet	86.0, 73.0, 70.1	42.5, 39.2, 35.2	75.0, 58.7, 55.2	**120.3**
		Ped., @0.7 IoU	Ped., @0.5 IoU	Ped., @0.3 IoU	
JRDB22 3D	PointPillars [18][2]	2.2	33.7	69.2	162.0
	PiFeNet [35][2]	4.6	39.8	70.7	138.5
	RPEA [36][2]	5.2	**45.4**	**74.4**	125.6
	PillarNet18S [20][1]	6.9	43.6	72.0	159.8
	PillarNet18 [20][1]	9.4	26.5	70.8	130.2
	SpikeCloudNet	3.6	29.6	71.3	**120.6**
	SpikeCloudNet$_{comb.}$	**10.0**	35.7	70.9	121.2

[1] Results reproduced and latency measured on NVIDIA RTX2080.
[2] Accuracy reported on KITTI/JRDB22 test split.

TABLE VI
SPARSITY, EFFECTIVE OPERATIONS AND MEMORY FOOTPRINTS EVALUATED ON KITTI VALIDATION SPLIT.

Backbone	Spars.	MACs	ACs	$M_{Sen.}$	M_{Feat}
PillarNetS	0.410	8.4e10	-	0.45MB	78.2MB
PillarNet	0.471	7.4e10	-	0.45MB	16.1MB
SCNet$_{\lambda_{reg}=0}$	0.852	3.3e9	1.7e10	**0.02MB**	5.46MB
SCNet$_{\lambda_{reg}=0.5}$	0.985	3.5e8	2.3e9	**0.02MB**	0.48MB
SCNet$_{\lambda_{reg}=1}$	0.992	9.6e7	1.1e9	**0.02MB**	**0.13MB**
SCNet$_{\lambda_{reg}=2}$	0.994	3.5e8	7.9e8	**0.02MB**	0.57MB
SCNet$_{\lambda_{reg}=5}$	**0.998**	**1.8e8**	**3.4e8**	**0.02MB**	0.28MB

TABLE VII
PERFORMANCE EVALUATION OF BACKBONES ON EDGE PLATFORMS.

Backbone	Hardware	$T_{Bb.}$ in ms	$T_{BbTot.}$ in ms	$E_{Dyn.}$ in mJ	EDP in mJs
PillarNetS	AGX Orin	42.4	142.4	61.7	8.79
PillarNet	AGX Orin	55.5	155.5	98.4	15.3
SCNet	AGX Orin	88.9	104.4	103.6	10.8
SCNet	Loihi2	**16.5**	**103.3**	**2.3**	**0.27**

edge hardware platforms for benchmarking: NVIDIA AGX Orin and Intel's neuromorphic research platform Loihi2 [32]. We set the AGX Orin in low power 15W mode and provide latencies for PillarNet [20] as a full scan reference and SpikeCloudNet. We use Intel's neuromorphic research processor Loihi2 to accelerate SpikeCloudNet's spiking backbone. Therefore we follow the recipe for neuromorphic inference described in section III-F2. In Table VII, we report the isolated backbone latencies $T_{Bb.}$ and the effective end-to-end backbone latencies $T_{BbTot.}$. The latter indicates the time after which the detection neck and head can start processing the backbone features. On the edge GPU the PillarNet backbone can be computed faster than the SpikeCloudNet backbone. Because of the streaming principle, $T_{BbTot.}$ is still lower than for the non-streaming PillarNet backbones. Running the SNN backbone on neuromorphic hardware gives additional latency gains. Even the pure backbone latency is lower than for PillarNet on edge GPU. Hence, we can assume latency improvements, also when the total scan time in the reduced range scenario is also smaller, e.g. below 30ms. We also evaluate the energy efficiency of running the backbone variants on the two platforms. We measure the energy on the AGX Orin with jetson-stats [38] and on Loihi2 with the probes provided in Lava [33]. For both cases, we calculate the reported dynamic energy $E_{Dyn.}$ from the difference of workload and idle power measurements. The SpikeCloudNet consumes slightly more dynamic energy than the reference backbones when executed on the AGX Orin. This is due to the iterative updates per scanline. As neuromorphic hardware is designed to accelerate these iterative sparse workloads, the spiking backbone can be executed 25 times more energy efficient than the most efficient PillarNetS backbone on AGX Orin. Additionally, we also provide the EDP as a joint metric of effective end-to-end backbone latencies $T_{BbTot.}$ and dynamic energy $E_{Dyn.}$. Table VII shows that the EDP of our SCNet on AGX Orin is similar to the reference PillarNet backbones on the same hardware. When ported on neuromorphic hardware, it outperforms the other setups by more than one order of magnitude.

V. CONCLUSION

In this work we showed that the presented methods for scanline-based LiDAR streaming and spiking processing lead to significant latency reduction of up to 10% compared to other

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

fast 3D object detectors. Additionally the bandwidth requirements from sensor to processor can be drastically reduced. We highlight that these advancements are achieved on standard GPU hardware which makes it easily applicable to a majority of robotic platforms. Further work on optimizing the streaming detector architecture and its inference on GPU could lead to even more benefits. Moving towards a neuromorphic processor can be benefical if there are additional energy constraints or faster LiDAR scanners to be supported.

REFERENCES

[1] Z. Liu *et al.*, "BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 2774–2781.

[2] R. Roriz, J. Cabral, and T. Gomes, "Automotive LiDAR Technology: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6282–6297, 2022.

[3] D. Wang, C. Watkins, and H. Xie, "MEMS Mirrors for LiDAR: A Review," *Micromachines*, vol. 11, p. 456, 4 2020.

[4] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for real-time object recognition," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928.

[5] H. Cai *et al.*, "Enable Deep Learning on Mobile Devices: Methods, Systems, and Applications," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, pp. 1 – 50, 2022.

[6] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-YOLO: An Euler-Region-Proposal for Real-Time 3D Object Detection on Point Clouds," in *Computer Vision – ECCV Workshops*, 2019, pp. 197–209.

[7] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85.

[8] W. Han *et al.*, "Streaming Object Detection for 3-D Point Clouds," in *European Conference on Computer Vision*, 2020.

[9] T. Beemelmanns *et al.*, "3D Point Cloud Compression with Recurrent Neural Network and Image Compression Methods," *IEEE Intelligent Vehicles Symposium (IV)*, pp. 345–351, 2022.

[10] Q. Chen, S. Vora, and O. Beijbom, "PolarStream: Streaming Object Detection and Segmentation with Polar Pillars," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[11] D. Frossard *et al.*, "StrObe: Streaming Object Detection from LiDAR Packets," in *Proceedings of the Conference on Robot Learning*, 2021, pp. 1174–1183.

[12] J. Eshraghian *et al.*, "Training Spiking Neural Networks Using Lessons From Deep Learning," *Proceedings of the IEEE*, vol. 111, pp. 1016–1054, 2021.

[13] F. Ottati *et al.*, "To Spike or Not to Spike: A Digital Hardware Perspective on Deep Learning Acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, pp. 1015–1025, 2023.

[14] M. Davies *et al.*, "Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[15] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[16] R. Martín-Martín *et al.*, "JRDB: A Dataset and Benchmark of Egocentric Robot Visual Perception of Humans in Built Environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, pp. 6748–6765, 2021.

[17] B. Li, "Vehicle Detection from 3D Lidar Using Fully Convolutional Network," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1513–1518.

[18] A. H. Lang *et al.*, "PointPillars: Fast Encoders for Object Detection From Point Clouds," *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12 689–12 697, 2018.

[19] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors*, vol. 18, no. 10, 2018.

[20] C. M. Guangsheng Shi, Ruifeng Li, "PillarNet: Real-Time and High-Performance Pillar-based 3D Object Detection," *ECCV*, 2022.

[21] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as Points," *ArXiv*, vol. abs/1904.07850, 2019.

[22] S. Zhou, Y. Chen, X. Li, and A. Sanyal, "Deep SCNN-Based Real-Time Object Detection for Self-Driving Vehicles Using LiDAR Temporal Data," *IEEE Access*, vol. 8, pp. 76 903–76 912, 2019.

[23] S. Mohapatra *et al.*, "SpikiLi: A Spiking Simulation of LiDAR based Real-time Object Detection for Autonomous Driving," *8th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*, pp. 1–5, 2022.

[24] A.-D. Vicol, B. Yin, and S. M. Bohté, "Real-time classification of LIDAR data using discrete-time Recurrent Spiking Neural Networks," in *International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–9.

[25] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Transactions on Circuits and Systems: Analog and Digital Signal Processing*, vol. 47, pp. 416–434, 2000.

[26] W. Fang *et al.*, "Incorporating Learnable Membrane Time Constant To Enhance Learning of Spiking Neural Networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 2661–2671.

[27] ——, "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence," *Science Advances*, vol. 9, 2023.

[28] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu, "Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection," *arXiv preprint arXiv:1908.09492*, 2019.

[29] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, pp. 51–63, 2019.

[30] Y. Yan, H. Chu, Y. Jin, Y. Huan, Z. Zou, and L. Zheng, "Backpropagation With Sparsity Regularization for Spiking Neural Network Learning," *Frontiers in Neuroscience*, vol. 16, 2022.

[31] J. Ansel *et al.*, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024, p. 929–947.

[32] S. Shrestha, J. Timcheck, P. Frady, L. E. Campos-Macias, and M. Davies, "Efficient Video and Audio Processing with Loihi 2," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 13 481–13 485, 2024.

[33] Intel, "Lava-NC," https://github.com/lava-nc, 2021, accessed: 2024-04-14.

[34] S. Shi *et al.*, "PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection," in *CVPR*, 2020.

[35] D.-T. Le, H. Shi, H. Rezatofighi, and J. Cai, "Accurate and Real-Time 3D Pedestrian Detection Using an Efficient Attentive Pillar Network," *IEEE Robotics and Automation Letters*, vol. 8, pp. 1159–1166, 2021.

[36] J. Guang, Z. Hu, S. Wu, Q. Zhang, and J. Liu, "RPEA: A Residual Path Network with Efficient Attention for 3D pedestrian detection from LiDAR point clouds," *Expert Systems with Applications*, vol. 249, p. 123497, 2024.

[37] Velodyne Acoustics Inc., "Velodyne's HDL-64E: A High Definition LiDAR Sensor For 3-D Applications," https://api.semanticscholar.org/CorpusID:26835230, 2009, accessed: 2025-01-14.

[38] Nvidia, "Jetson Stats," https://github.com/rbonghi/jetson_stats, 2023, accessed: 2024-11-14.