EmPointMovSeg: Sparse Tensor-Based Moving-Object Segmentation in 3-D LiDAR Point Clouds for Autonomous Driving-Embedded System

Zhijian He, Xueli Fan, Yun Peng, Zhaoyan Shen, Jianhao Jiao, and Ming Liu, *Senior Member, IEEE*

***Abstract*—Object segmentation is a per-pixel label prediction task that targets at providing context analysis for autonomous driving. Moving-object segmentation (MOS) serves as a subbranch of object segmentation, targeting to separating the surrounding objects into binary options: dynamic and static. MOS is vital for the safety-critical task in autonomous driving because dynamic objects are often a true potential threat to self-driving cars compared to static ones. Current methods typically address the MOS problem as a category feature to label the mapping task, which is not rational in reality. For example, a parking car should be considered as static instead of a moving-object category. There is a little systematic theory to differentiate object moving characteristics from nonmoving characteristics in MOS. Furthermore, restricted by limited resources in the embedded system, MOS is often in an offline manner due to huge computational requirements. An online and low computational cost MOS is an urgent demand for the practical safety-critical mission which takes immediate reaction as compulsory. In this article, we propose EmPointMovSeg, an efficient and practical 3-D LiDAR MOS solution for autonomous driving. Leveraging the power of the well-adapted autoregressive system identification (AR-SI) theory, EmPointMovSeg theoretically explains the moving-object feature in large-scale 3-D LiDAR semantic segmentation. An end-to-end sparse tensor-based CNN which balances segmentation accuracy and online process ability is proposed. We construct our experiment on both representative dataset benchmarks and practical embedded systems. The evaluation result shows the effectiveness and accuracy of our proposed solution, conquering the bottleneck in the online large-scale 3-D LiDAR semantic segmentation.**

***Index Terms*—Deep learning, embedded system, LiDAR moving-object segmentation.**

Manuscript received 10 January 2022; revised 22 March 2022; accepted 26 April 2022. Date of publication 9 May 2022; date of current version 22 December 2022. This work was supported in part by the Zhongshan Municipal Science and Technology Bureau Fund under Project ZSST21EG06, and in part by the Collaborative Research Fund by Research Grants Council Hong Kong under Project C4063-18G. The work of Ming Liu was supported by the Department of Science and Technology of Guangdong Province Fund under Project GDST20EG54. This article was recommended by Associate Editor N. K. Jha. *(Corresponding author: Ming Liu.)*

Zhijian He, Yun Peng, Jianhao Jiao, and Ming Liu are with the RAM Lab of Robotics Institute, The Hong Kong University of Science and Technology, Hong Kong, SAR, China (e-mail: eehezhijian@ust.hk; pyun@connect.ust.hk; jjiao@connect.ust.hk; eelium@ust.hk).

Xueli Fan is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, SAR, China (e-mail: csxfan@comp.polyu.edu.hk).

Zhaoyan Shen is with the School of Computer Science and Technology, Shandong University, Qingdao 266200, China (e-mail: shenzhaoyan@sdu.edu.cn).

Digital Object Identifier 10.1109/TCAD.2022.3172031

I. Introduction

OBJECT segmentation serves as an important role in autonomous driving safety. Via predicting the per-pixel label in a driving scene, the object segmentation algorithm is able to provide comprehensive context information to the system, which benefits the perception result so that path planning can be more accurate. For example, by detecting lane, traffic sign, etc., the autonomous driving system is able to get an excellent description of the environment so that smart planning decision is obtained. However, for the safety-critical mission in autonomous driving, the potential threat mainly lies on pedestrian, cars, or trucks which have moving characteristics in common. Less danger is caused by static objects, such as fence, trees, traffic signs, etc. This phenomenon generates a subresearch topic in object segmentation, the so-called moving-object segmentation. Although many solutions exist in multicategory object segmentation, the moving-object segmentation (MOS) domain, which requires the heavy computational expense and the knowledge of distinguishing dynamic objects from static ones [1]–[4], is still a bottleneck in autonomous driving safety object segmentation.

The majority of current segmentation methods [5], [6], [7] addresses this task as a supervised-learning per-pixel classification problem. Typically, a U-Net [8] like CNN takes in a category-specific feature and minimizes the network loss between prediction output and annotated label. For example, a cluster of 3-D LiDAR data containing trunks and branch feature is considered as a tree. After finishing the capture process, these methods adopt an encode–decode pattern to train a context understanding network, resulting in high accuracy multiple class object segmentation. Unlike the majority of current existing segmentation methods which inputs raw 3-D LiDAR data and outputs multiple label accordingly, MOS emphasizes a binary segmentation task that figures out moving objects from nonmoving objects in the autonomous driving scene. MOS considers the moving objects as the root cause of many traffic accidents so that it cannot be constructed as a simple binary segmentation problem. For example, in Fig. 1, the car (yellow box) parking at the roadside, opposite to the car (red box) driving on the road, although belonging to a moving-object category, should not be considered as moving semantically. Only the red mask in the right subpicture linking to the red box in the left subpicture is a correct moving-object segmentation.

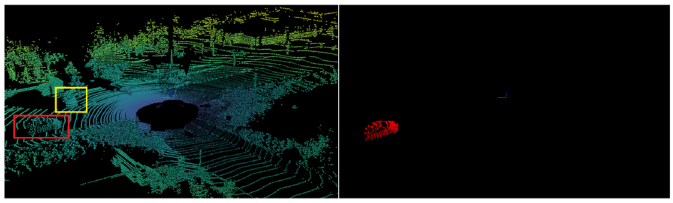

Fig. 1. Drawbacks in traditional classification-like 3-D LiDAR semantic segmentation.

As a result, an extra feature that contains the information of differentiating moving objects from static ones needs to be added into deep-learning training. The essential of this feature can be regarded as a changing position in sequential data. As a result, finding temporal geometric change is a vital solution to the MOS. However, the varying density in large-scale LiDAR data formulates the difficulty in exploring the temporal geometric relationship in reality. This feature of the point cloud limits the online LiDAR segmentation application for the reason that both the temporal [1] and the spacial information [5] have to be considered at the same time, which generates two NP-hard problems.

1) Calculating too many features in runtime requires expensive computational resources at the same time, especially in case that online moving segmentation is compulsory for the safety-critical mission.
2) The temporal feature should be theoretically explained so that the result of differentiating the binary mask in MOS can be guaranteed.

For the first problem, current methods prepare preloading relative pose information and raw point cloud in advance. Via powerful GPU cloud computing or excellent graphic hardware, 3-D LiDAR data can be manipulated to achieve accurate segmentation results [6], [9], [10], which is not the real environment settings in the practical autonomous driving-embedded system. Although polar shape [5] or other rational spacial structure [10] in expressing point cloud is explored, current methods emphasize the effect of raising intersection over union (IoU) instead of considering both the efficiency and accuracy in segmentation. Due to the large batch size during the inference phase, accurate IoU can be achieved only in an offline manner, which is useless given no reaction possibility is considered in reality. In the safety-critical compulsory scenario, the true positive dynamic characteristics of an object are more important than high category accuracy. Hence, efficient online accurate MOS is the goal in guaranteeing autonomous driving safety. In this article, in virtue of sparse tensor and sparse convolution, we overcome the varying density feature via avoiding the empty space of LiDAR data to operate in convolution so that segmentation inference time reduces. Furthermore, understanding the shape of a moving object is enough for practical semantic segmentation comparing to describe the point cloud densely. These two novel operations shrink the inference time to make online MOS possible in an embedded system.

For the second problem, the widely used encode–decode architecture CNN [8] is used as an efficient and accurate solution network in semantic segmentation. The well-picked temporal feature can construct powerful input to the network to improve moving-object segmentation significantly [1]. However, current methods [1], [11]–[14] just estimate relative object states intuitively to distinguish dynamic object from static ones. In this article, we utilize the well-adapted autoregressive system identification (AR-SI) theory [15] to propose a systematic approach to discover the true dynamic characteristics of an object. Via AR-SI oracle judgment, a dynamic object in an autonomous driving scene can be confirmed. This so-called AR-SI filter constructs a temporal feature serving as the following CNN input. Combining the special feature in the raw point cloud and this generated temporal feature, we map the label for segmentation training to generate an accurate IoU result. Targeting at the binary mask of discovering the moving object in the real scene, we establish the AR-SI theory into feature selection not only to describe the moving character of dynamic objects but also to block the perturbation from static objects which is totally a blind area in traditional methods [16]–[18] using geometric feature only.

In this article, we construct the temporal and the spacial feature to input to an encode–decode architecture CNN and use the sparse network architecture to predict the moving binary mask (details of the overall solution are illustrated in Section III-B). Specifically, the contributions of this work are listed as follows.

1) We theoretically explain the essence of the model that distinguishes static objects from dynamic ones in autonomous driving. Leveraging the power of 3-D LiDAR residual depth in temporal representation, we propose a novel AR-SI-based feature to improve the encode–decode architecture-based CNN prediction significantly. Combining both the filtered temporal feature and the geometric feature, the CNN prediction is able to detect the true dynamic object ignoring the perturbation from the same category.
2) We propose sparse tensor and sparse convolution to handle the unconstructed raw LiDAR point cloud, which shrinks the gap between segmentation algorithm in the powerful GPU only and its practical use in the autonomous driving-embedded system.
3) We evaluate our approach on the widely used large-scale LiDAR dataset SemanticKITTI to prove the segmentation performance. Then, we deploy our method on a real autonomous driving-embedded system to check its practical effect.

The remainder of this article is organized as follows. Section II gives the background and motivation of conquering traditional semantic segmentation drawbacks in autonomous driving-embedded systems. Section III presents the details of using AS-RI theory to distinguish moving and nonmoving objects. Using this theoretical analysis result, we illustrate our network details in Section IV. In Section V, we present experimental results. Finally, via comparing the related work in Section VI, Section VII provides the conclusion.

II. Background and Motivation

In this section, we first give a brief overview of LiDAR semantic segmentation. The basic strategy on current LiDAR semantic segmentation is then illustrated. Finally, we present a

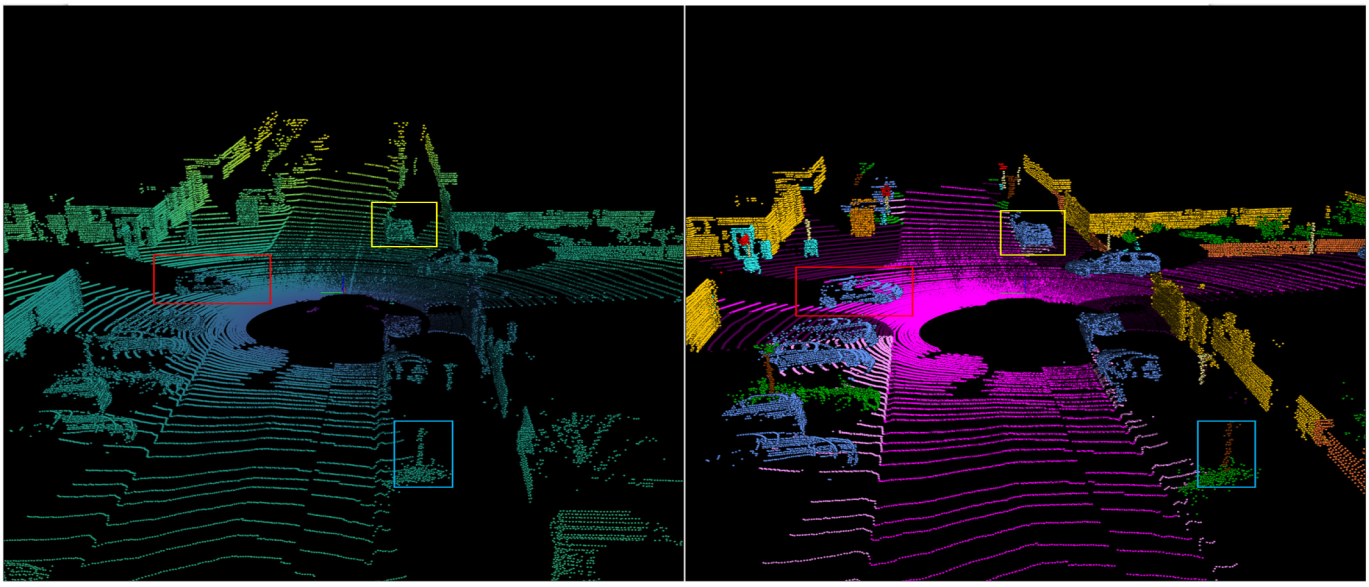

Fig. 2. 3-D LiDAR semantic segmentation. The left picture is a raw LiDAR representation while the right one is indicated in an annotated manner.

motivation example for the online 3-D LiDAR moving-object segmentation on the resource-limited-embedded system.

A. Large-Scale 3-D LiDAR Point Cloud Segmentation Task

The autonomous driving core technologies can be mainly divided into two classes [19]: 1) vision based and 2) LiDAR based. LiDAR sensor measures the time gap between the reflected light sender and the receiver, which can be used to construct 3-D object points representation via multiplying LiDAR ware length. Denoted as $(x_i, y_i, z_i, \text{intensity}_i)$, raw LiDAR data contains reflected light position and its intensity, where i means sequence index. As Fig. 2 shows, the left image is the raw point cloud that displays spacial objects. However, this raw data scan cannot satisfy perception requirements for the reason that no semantic or instance information is provided. For example, the car in the red box is meaningless to the system unless a pixel label is added. As the right image of Fig. 2 shows, object labeling made by annotation tool or handcraft provides scene understandable message, which can be used for either training or evaluation in the segmentation task.

Current segmentation methods, such as PointNet [16] input scan of dense 3-D LiDAR data to a CNN to construct an end-to-end feature-to-label supervised training. The network model parameters are generated once the loss between the point prediction and the ground truth is satisfied. The CNN plays the role of mapping geometric point cloud correlationship to the annotated label. The inference is performed using new captured dense LiDAR data to produce a prediction mask/label. Typically, this task contains tens of category labels for semantic segmentation, which is very similar to the deep-learning-based classification problem [20].

B. Moving-Object Segmentation

Comparing to all classes of semantic segmentation, moving-object segmentation indicates the task of distinguishing a static object from dynamic ones in large-scale sequential 3-D LiDAR perception. This process typically merges the original labels into three classes: 1) moving; 2) nonmoving; and 3) unlabeled [1]. For example, car and bicyclist should be reorganized as moving labels while fence and trunk are considered as nonmoving. The blue bounding box in Fig. 2 indicates a tree labeled as nonmoving while the red bounding box means a car labeled as moving.

The input to the evaluated method is a list of coordinates of 3-D points along with their remission. Each method should then output a label for each point of a scan. To assess the network performance, IoU over moving and nonmoving parts of the environment are investigated [21]. Higher IoU means the performance of the network is better.

C. Corner Case in MOS

The current majority of methods in pure LiDAR data segmentation uses a feature-to-label mapping principal to train a CNN [5], [16], [22]. However, as Fig. 2 shows, the parking car in the yellow bounding box is a static object while the red bounding box car driving in the road stands for a real moving object. Although both of them are labeled as the car in the ground truth, in moving-object segmentation, we should not treat it using the category feature only. Misleading by wrong prediction label causes confusion in driving safety reaction because the parking car is actually harmless. More importantly, if an incorrect static label is given by the system, a road accident may be triggered because the system treat potential danger as a static safe condition.

D. Motivation

Static objects are relatively safe compared to moving ones, for example, the fence is not able to cause damage unless all the sensors on the car are disabled. In other words, moving-object judgment plays a vital role in autonomous driving

safety. This importance of combining unsatisfied performance in existing moving-object segmentation approaches motivates us to design our systematic solution with the following goals.

1) *Efficient:* A systematic moving-object segmentation method should handle the LiDAR batch fast enough to achieve online computation. We achieve this goal by leveraging the power of sparse tensor and sparse convolution.
2) *Accurate:* After considering the solution efficiency, segmentation design should be accurate not only to distinguish static and dynamic objects that belong to different classes but also should identify true movable objects belonging to the same moving class. In this work, we use AR-SI theory to infer temporal feature which is added into a traditional spatial feature for the CNN input.
3) *Practical:* The whole methodology should be able to be deployed in an embedded system to prove its practical use. The system should be deployed in a real autonomous driving-embedded system to meet the requirements of accurate moving-object segmentation while incurring a low computational cost. Online performance is critical and useful for guaranteeing autonomous driving safety.

III. Related Work

A. Semantic Segmentation

Semantic segmentation is a task that assigns each pixel a category label in an image or point cloud. Semantic segmentation appears in both vision and LiDAR domains and gains much attention in recent years. Fully convolutional networks (FCNs) [23] replace fully connected layers in image classification task with convolution layers, which turns per-pixel labeling possible in 2015. U-Net [8] creates the Encoder-Decoder architecture which constructs the cornerstone for the vision semantic segmentation task. Badrinarayanan *et al.* [22] improved this architecture to develop it on both road scenes and SUN RGB-D indoor scene segmentation tasks [24]. Leveraged the power of the Encoder-Decoder architecture, semantic segmentation in large-scale dynamic 3-D LiDAR point cloud has been successfully applied. Majority of the 3-D LiDAR semantic segmentation target at mapping multiple categories label [5], [6] to each pixel on a road scene so that the context of the open road can be understood. From semantic segmentation to panoptic segmentation [25], [26], polar coordinate [5] and other project coordinate [27] have been applied to organize the point cloud coordinate for calculating the according label in CNN. Dense point cloud approaches, such as [1], [6], and [28] have gained great achievement on multiple categories of semantic segmentation. However, in the aspect of a safety-critical mission, autonomous driving emphasizes the potential threat in a true moving object other than a typical nonmoving object. The most important breakthrough is [1], which targets at distinguishing a true dynamic object from the false-positive dynamic object using multiple categories of semantic segmentation methods. For the sparse tensor approach, PointMoSeg [29] combines temporal and spatial information in 3-D LiDAR point cloud segmentation, which is mostly relative to our approach. However, it is not evaluated on an embedded platform to check its practical performance.

B. Moving-Object Segmentation Task

Moving-object segmentation, which is slightly different from traditional multiple label mapping semantic segmentation [27], [30], aims to discover the real moving objects from nonmoving ones [1]. This task can be divided into two kinds of approaches: 1) Map-use and 2) Map-free. Map-use approaches in 3-D LiDAR semantic segmentation typically use two steps to manage moving-object segmentation. First, sensor data are captured offline containing ego motion and LiDAR data. A prebuilt map is generated to use time-consuming voxel or grid base methods in order to separate dynamic objects from static using real-time captured data in the second step [31], [32]. These kinds of approaches require a clean prebuilt map and, therefore, cannot be used in online semantic segmentation. Map-free approaches indicate online methods capture real-time LiDAR and ego-motion data to perform the per-pixel labeling task. Ruchti and Burgard [33] used probability to predict movable objects. Dewan *et al.* [34] proposed a method based on a rigid body using a LiDAR point cloud. It formulates the problem as an energy minimization problem that estimates motion vectors for rigid bodies. Dewan and Burgard [35] proposed a deep convolutional neural network (DCNN) for semantic segmentation of a LiDAR scan. It proposes the Bayes filter-based method to make the predictions from the DCNN semantic state temporally consistent. These kinds of methods do not mention about their online segmentation performance, which is inevitable in practical safety-critical missions. Bogoslavskyi and Stachniss [36] presented an effective method that segments the 3-D data in a range image representation via removing the ground from the scan. These existing approaches do not theoretically explain the rational math model of moving-object segmentation. Instead, they make intuitive approaches in pose estimation or temporal 3-D LiDAR scan sequence to solve moving-object segmentation. AR-SI theory [15] is well-established math in control-CPS, which can be served as the oracle not only in moving/nonmoving binary judgment but also can be used in checking the moving track quality. This inspires us to systematically explain the binary feature in the input tensor of the CNN, which is novel in moving-object segmentation.

A context analysis task is often equal to a high workload computational task. Projected image approaches consume a huge amount of computational time, although of high accuracy, and are not practical in autonomous driving. Thanks to the power of sparse tensor/convolution, we can perform an online approach in an embedded system. Similar to [1], we train the network using binary masks to achieve an end-to-end fashion. However, we implement the CNN network using the sparse tensor-based manner while Chen *et al.* managed it in a dense tensor way.

IV. Solution Theory

Moving-object segmentation indicates an environment semantic context analysis in autonomous driving. Aiming to assist safety-critical reaction, this task addresses the problem

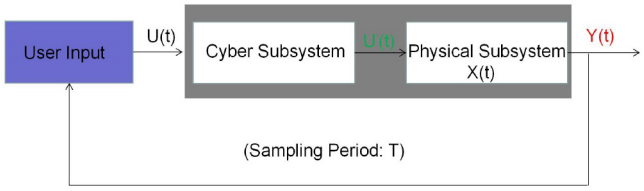

Fig. 3. Typical control-CPS architecture.

of distinguishing the dynamic object from static ones around the vehicle. In this section, we detail the math theory in discovering a moving object, which figures out the essential math model not similar to the majority of label-based deep-learning methods. Existing methods use sequential LiDAR scan as moving-object segmentation temporal feature but none provides a theoretical explanation.

A. Math Model in Distinguishing Moving From Nonmoving Objects Using AR-SI Theory

We treat each object in the autonomous driving scene as a control-CPS system. For each control-CPS system, we regard a static object as no control input while dynamic ones as the opposite [37]. For example, trees at the roadside can be considered as a typical control-CPS without user impulse, and cars can be treated as control-CPS with regular user pedal input. A classical control-CPS consists of a user input module, cyber subsystem, and physical subsystem as Fig. 3 shows.

Generally, the user control input $U(t) \in R^q$ at time $t \in [0, +\infty)$ is generated in a regular period to pass onto the cyber subsystem [15]. (For example, the car driver operates the pedal to the wheel at a preset period. If no user signal is captured, 0 is taken as the input of the cyber subsystem.) The cyber subsystem inquires the positioning result from the navigation subsystem and then calculates the error $U^{'}(t)$ between the reference point and current position. $U^{'}(t)$ formulates the final impulse to the physical subsystem (also called "physical plant"). The physical subsystem combines current state $X(t)$ and the impulse to produce the trajectory $Y(t)$ of the whole system. The user adjusts its input according to the physical plant trajectory during a sampling period T.

Specifically, if we regard the gray area of Fig. 3 as a black box, whose input is $U_i \in R^q$ and output is $Y_i \in R^m$, then AR-SI typically models the relationship between U_i and Y_i $(\forall i = p, p+1, \ldots,)$ as [15]

$$Y_i = \left(\sum_{j=1}^{p} A_j Y_{i-j} \right) + BU_i + \xi_i \tag{1}$$

where Y_i stands for the physical subsystem position at time step i. p means the time steps tracking back Y_i, and the A_j is the corresponding weight. U_i is the user input at time step i, and B is the weight parameter of user input. ξ_i indicates the SI error item. At time step i, last time step $i-1$ consecutive plant states $(Y_{i-p-1}, \ldots, Y_{i-2}, Y_{i-1})$ become available. The according parameters $A_1, A_2, \ldots, A_p$ and B of time step $i-1$ in (1) are optimized using AR-SI theory until the runtime accumulated SI error energy is minimized. The AR-SI model uses these amount of optimized parameters and available plant states $(Y_{i-p}, \ldots, Y_{i-1})$ to predict Y_i with $\hat{Y}_i$ [15]

$$\hat{Y}_i = \left(\sum_{j=1}^{p} A_j^{(*,i)} Y_{i-j} \right) + B^{(*,i)} U_i \tag{2}$$

When Y_i is available, the AR-SI prediction error becomes

$$e_i = \hat{Y}_i - Y_i. \tag{3}$$

A static object in autonomous driving scene can be considered as no user input, and its consecutive plant states are the same. This model can be calculated as follows:

$$Y_i = \left(\sum_{j=1}^{p} A_j Y_{i-j} \right) + \xi_i \tag{4}$$

where A_j equals $1/p$. However, there exists certain sensor noise which forces the sum of the averaged Y_{i-j} cannot perfectly be equal to Y_i. As a result, (3) is optimized as follows:

$$e_i = \left\| \hat{Y}_i - Y_i \right\| < \epsilon. \tag{5}$$

The threshold value ϵ can be inferred through Monte Carlo [38] sampling. If the object trajectory approximation error using the AS-RI model is lower than ϵ, we are able to predict it as nonmoving and *vice versa*.

B. Proposed Solution

Using the math model in the last section, we combine the advantages from two sides: 1) spatial feature as the majority of feature-to-label mapping segmentation proposed and 2) temporal feature as AR-SI math describes. The overall solution is as Fig. 4 shown.

Each loop in the system takes in a raw LiDAR point cloud containing spatial reflected information and remission measurement. A sequence of LiDAR frames, from current time step T_{N-1} to the frame back N steps T_0, generates a temporal vector using spherical coordinate projection. The temporal vector contains P residual depth (details explained in Section IV). This set of residual depth, combing spatial raw vector from current time step T_{N-1}, will be pass into the AR-SI filter. The AR-SI filter uses (1)–(5) to check whether it is a static object or not. The temporal vector is turned into all zero in case of AR-SI filter error is lower than ϵ, which means AR-SI considers it as nonmoving. This operation benefits the CNN training because the nonzero trajectory value is actual noise if the object is confirmed as static, especially in case that the training problem is binary. In order to distinguish different static objects, the spatial vector maintains the same for CNN training. In the aspect of dynamic ones, the temporal vector maintains the residual depth value to concatenate with the spatial vector to formulate the dense input of the sparse tensor module.

Then, dense input tensors are voxelized to construct the sparse tensor for the CNN considering fast inference requirements in the embedded system. Sparse tensor indices each voxel with coordinate (c_x^N, c_y^N, c_z^N) with b^N and attaches the according feature (f_x^N, f_y^N, f_z^N). Feature details will be illustrated in Section IV. The feature tensor is passed into the

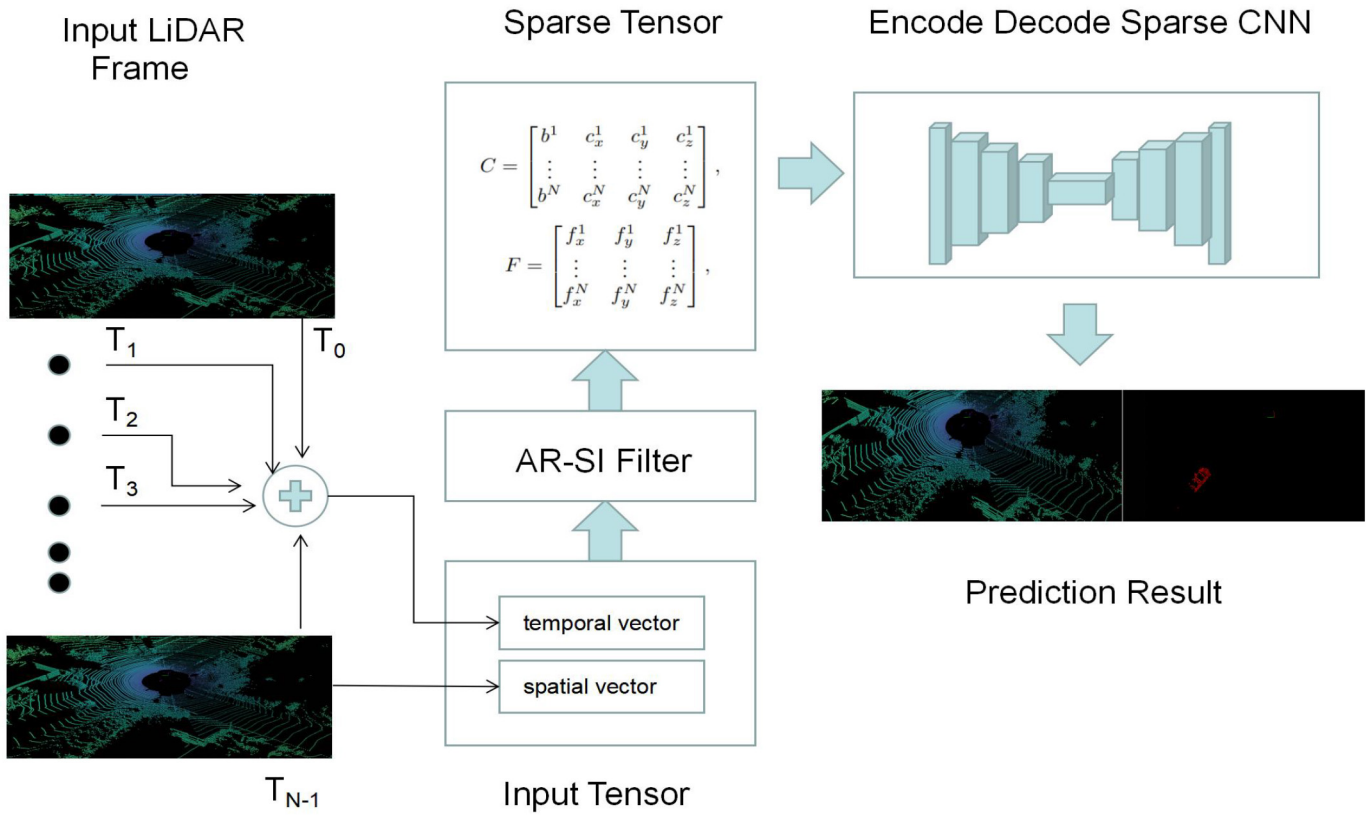

Fig. 4. Overview of our proposed method.

encode–decode sparse CNN to construct the sparse convolution. By dividing the label into three categories only: 1) unlabel; 2) dynamic; and 3) static, we train the CNN model prediction to map the label accordingly. The prediction result is shown via discovering dynamic objects in the red mask, which is the vital potential threat to autonomous driving safety.

V. Sparse Tensor-Based Moving-Object Segmentation

A. Sparse Tensor and Sparse Convolution

Moving-object segmentation leverages the power of captured point cloud geometric information to extract low-level features. The features are used to fill into the learning network in the following step to predict the segmentation mask. However, typical methods such as PointNet [16] operate the original unconstructed data directly to fetch ready-for-train geometric features, which consumes huge computation resources. In this work, we adapt sparse tensor to improve computational efficiency to make the moving-object segmentation practical in an embedded system.

The sparse tensor uses a voxel coordinate matrix C to index an associated feature matrix F. The first step is to voxelize the point cloud and make one voxel contain only one point cloud, which generates the according voxel coordinate C. This step takes the advantage of removing redundant points in a same voxel, which avoids empty 3-D voxel computation so that massive memory footprint generation can be avoided. The second step is to calculate the associated feature matrix F according to the generated index coordinate C

$$C = \begin{bmatrix} b^1 & c_x^1 & c_y^1 & c_z^1 \\ \vdots & \vdots & \vdots & \vdots \\ b^N & c_x^N & c_y^N & c_z^N \end{bmatrix}, \quad F = \begin{bmatrix} f_x^1 & f_y^1 & f_z^1 \\ \vdots & \vdots & \vdots \\ f_x^N & f_y^N & f_z^N \end{bmatrix} \tag{6}$$

where for point i, $\{c_x^i, c_y^i, c_z^i\} \in \mathbb{Z}^3$ is the voxelized integer coordinate. The b^i is the batch index attached to the coordinate $\{c_x^i, c_y^i, c_z^i\}$. $\{f_x^i, f_y^i, f_z^i\} \in \mathbb{R}^3$ is the float-type coordinate captured by a 3-D LiDAR, $i \in [1, N]$, and N is the number of points after quantization, which is determined by the voxel size, $N \leq N_o$, where N_o is the original number of points before quantization.

The sparse convolution [28] inputs a sparse tensor and also outputs a sparse tensor with varying batch length. Specifically, it first generates the coordinate matrix C^{out} for the output sparse tensor from the given input coordinate matrix (details are described in [28]). Then, it calculates the feature vector $\mathbf{f}_{\mathbf{c}}^{\text{out}}$ for an output coordinate $\mathbf{c}$ with the formula [29]

$$\mathbf{f}_{\mathbf{c}}^{\text{out}} = \sum_{\mathbf{s} \in \mathcal{N}(\mathbf{c}, K)} W_{\mathbf{s}} \mathbf{f}_{\mathbf{c}+\mathbf{s}}^{\text{in}}, \quad \mathbf{f}_{\mathbf{c}}^{\text{out}} \in F^{\text{out}}, \quad \mathbf{c} \in C^{\text{out}} \tag{7}$$

where $\mathbf{s}$ is the offset to find the corresponding input coordinates, they are centered in $\mathbf{c}$ position and covered by the kernel size K, which is denoted as $\mathcal{N}(\mathbf{c}, K)$. $\mathbf{f}_{\mathbf{c}+\mathbf{s}}^{\text{in}}$ means the input feature vector at the input coordinate $\mathbf{c} + \mathbf{s}$. $W_{\mathbf{s}}$ represents the coefficient parameter, which is to be learned through the

training process. The output sparse tensor, coordinate matrix C^{out}, and feature matrix F^{out} can be generated via training.

B. Spacial Residual Depth

Inspired by the observation and conclusion in [1] and [11], residual depth projected in spherical coordinates is the vital factor in distinguishing moving and nonmoving objects. We take residual depth $d^i_{(u,v)}$ [1] as the trajectory symbol in (5), so our AR-SI filter expression can be calculated as follows:

$$e_i = \left\| \hat{d^i_{(u,v)}} - d^i_{(u,v)} \right\|. \tag{8}$$

If the residual depth error is less than the calculated filter value ($e_i < \epsilon$), we consider it as static and turn all the values in the temporal vector to zero. In (8), $d^i_{(u,v)}$ is the residual depth in position (u, v) of the projected spherical coordinates. The $\hat{d^i_{(u,v)}}$ means using residual depth data in time slice $i-1$ and the data back P steps in (4). If an object is static, the sum of its depth variance within this slicing temporal window should be less than ϵ. Considering that noise exists in the sensor, the ϵ can use Monte Carlo sampling to calculate.

Guided by the goal from (8), we require the value of the residual depth $d^i_{(u,v)}$ in time sequence $i+1-P$ to i. We use a range projection of LiDAR data to generate this residual depth d. The strategy is to convert each LiDAR point at time i, $P = (x_i, y_i, z_i)$ to time slice $i+1-P$ coordinate. Two range information $r^i_{(u,v)}$ and $r'^i_{(u,v)}$ exists in the same projected point (u, v) which can be mapped to LiDAR point (x, y, z) in 3-D space. The depth variance at the same point can be measured as [1]

$$d^i_{(u,v)} = \frac{\left\| r^i_{(u,v)} - r'^i_{(u,v)} \right\|}{r^i_{(u,v)}}. \tag{9}$$

The range information $r'^i_{(u,v)}$ is calculated as the following:

$$p_{(u,v)}{}^i = T^{i+1-P \to i} \cdot p^{i+1-P}_{(u,v)} \tag{10}$$

$$r'^i_{(u,v)} = \sqrt{p_x^{i,(u,v)^2} + p_y^{i,(u,v)^2} + p_z^{i,(u,v)^2}} \tag{11}$$

where $T^{i+1-P \to i}$ means the pose transformed from time step $i+1-P$ to i. We describe a consecutive sequence from time 0 to N of pose information as $T^1_0, \ldots, T^N_{N-1}$, where T^1_0 is the pose transforming from time step 0 to 1. To benefit from the continued product character of homogeneous coordinates, we represent the LiDAR data as $p_i = (x_i, y_i, z_i, 1)$.

The last step is to associate the 3-D LiDAR point to 2-D projected point, saying: $R^3 \to R^2$ to spherical coordinates, and finally to image coordinates

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\left[1 - \arctan(y,x)\pi^{-1}\right]\omega \\ \left[1 - (\arcsin(zr^{-1}) + f_{\text{down}})f^{-1}\right]h \end{pmatrix} \tag{12}$$

where (u, v) are image coordinates representation mapping the 3-D LiDAR point (x, y, z), and f defines the sensor vertical field of view as $f = |f_{\text{down}}| + |f_{\text{up}}|$. Following the work of [39], we considered a 3-D 360° field of view during the projection process. The h, ω stands for the height and width of the desired range image. The calculated coordinates (u, v) serves as the index in a hash map which contains the information of each point cloud data x, y, and z coordinates, its remission e, and its range information r.

C. Network Architecture

We adopt the most promising Encoder-Decoder architecture to our sparse tensor-based network design. Inspired by [27], the overall architecture of the proposed network is shown as Fig. 5. The input to the proposed network is a sparse tensor containing the spatial information from the raw 3-D LiDAR data and the temporal information from the residual image. Specially, each pixel (u, v) of the projected range image contains a vector $(x, y, z, r, e, d_0, d_1, \ldots, d_P)$ [1], where (x, y, z, r) is the spatial information of the raw 3-D LiDAR point cloud. Range r of each point can be calculated as $r = \sqrt{x^2 + y^2 + z^2}$.

The overall network can be divided into four modules, namely, the contextual enhance module, encoder module, decoder module, and prediction logit module. The contextual enhance module plays the role of fusing a local feature with a larger one at the beginning of the network. However, due to the varying length of the sparse tensor and multiple dimensional input (feature and coordinate), we use MinkowskiEngine [40] to handle the batch for each training.

The contextual convolution follows the pattern in [27] which has one $1*1$ and two $3*3$ kernels with dilation rate of $(1, 2)$. Concatenating two context blocks to generate an N*32 tensor serves as the input of the encoder module. The purpose of this convolution is to make the $1*1$ sparse convolution captures the local information while $3*3$ gets the global feature. Via concatenating these two features together, we get a $(N, 32)$ sparse tensor that captures context information of the raw 3-D LiDAR point cloud.

The encoder module contains four residual blocks for downsampling sparse convolution. Each residual block has three sparse convolutions, ReLU, and BatchNorm layers. All these basic blocks are revised using MinkowskiEngine to fit the sparse convolution requirement. These residual blocks' output concatenate together to formulate output using residual principle as [41]. One dropout and one pooling layer attach to the end of the residual block to avoid the overfitting problem.

The decoder module has a sequence of upsampling blocks similar to [27]. Each upsample block has three sparse convolution, ReLU, and BatchNorm layers. Three upsampling block outputs are concatenated together. One dropout layer follows this result at the end of the decoder module. The logit module maps the upsampling output to the point cloud label to generate the final prediction. F block is a tool in MinkowskiEngine used to convert output sparse tensor to normal tensor for the segmentation result.

D. Loss Function

The class label imbalance problem exists in both typical dataset and practical scene. For example, in the campus where we collect data, static objects are the majority of cases comparing to dynamic ones. As a result, training becomes useless if one sequence overfits static ones and ignores dynamic cases, which yields poor performance in network inference.

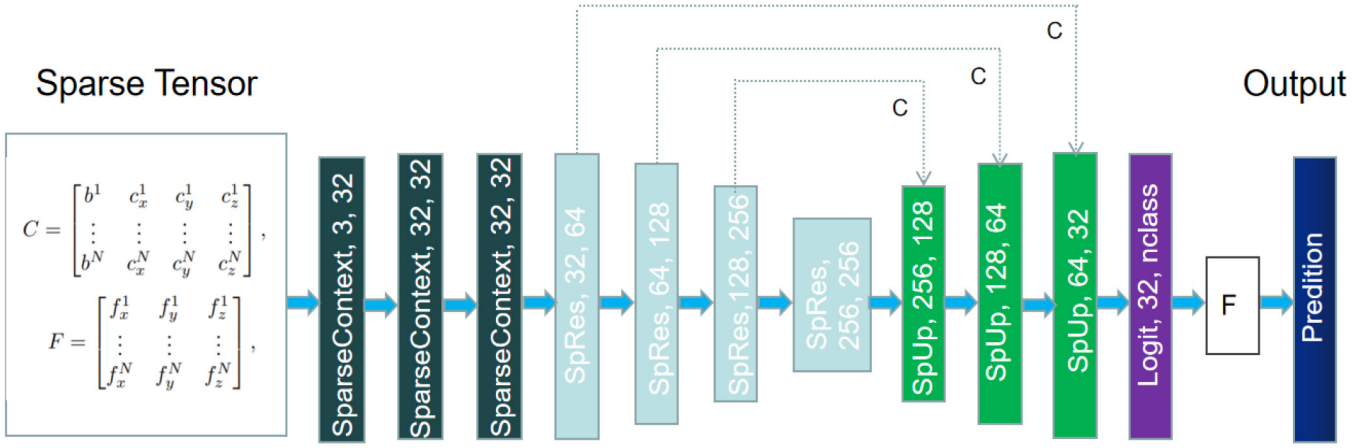

Fig. 5. Architecture of EmPointMovSeg: sparse tensor-based encode–decode sparse convolution network.

Therefore, we propose a strategy to cope with this class labeling imbalance problem similar to [27]: calculating loss by emphasizing the effect of under-presented classes via taking its appear frequency into account. The loss formula is expressed as follows:

$$\mathcal{L}_{wce}(y,\hat{y}) = -\sum_{i} a_i p(y_i)\log\big(p(\hat{y}_i)\big) \quad \text{with} \quad a_i = 1/\sqrt{f_i} \tag{13}$$

where $p(y_i)$ stands for the probability of label y_i and $p(\hat{y}_i)$ means the probability of prediction label of y_i. f_i indicates the frequency. i is the ith class which contains only three class in moving-object segmentation. This loss function takes the class imbalance problem into consideration to avoid the partially overfitting certain class phenomenon.

VI. Experimental Evaluation

This article targets at developing a practical moving-object segmentation solution in 3-D LiDAR scans. We construct our first experiment on a most representative dataset to prove the efficiency of our proposed method. The second evaluation is performed on a practical-embedded system platform to check the real application effect. The following evaluation results show that: 1) our proposed method achieves comparative segmentation performance while maintaining a low computational cost and 2) the relatively comparative low computational cost leads to practical moving-object segmentation success, which is able to apply the proposed online segmentation method in safety-critical tasks.

A. Evaluation Based on Dataset

To fulfill the online segmentation performance in an embedded system, the first experiment in this section is to support our claim of achieving comparative segmentation performance while a low inference time is required.

We first test all the methods on a powerful server equipped with:

1) Intel Xeon Gold 5118 CPU;
2) sytem memory up to 128 GB;
3) NVIDIA GeForce RTX 3090 Graphic Card.

The dataset we choose should fulfill the harsh autonomous driving requirements comparing to the traditional dataset just for a single object segment.

1) Large-scale sequential 3-D LiDAR-based data, including cloud point spacial and reflection information.
2) Containing ground truth in dynamic traffic participants with distinct classes, which can reason our model rationality.

We choose SemanticKITTI [21] as our testing dataset. Besides fulfilling the above requirements, SemanticKITTI provides LiDAR-based pose information so that odometry, object detection, tracking, semantic segmentation, panoptic segmentation, and scene completion tasks are proposed as open challenge issues for dataset users. SemanticKITTI is currently the most representative benchmark for moving-object segmentation.

The SemanticKITTI dataset contains 22 dense LiDAR individual scans. For sequences $00-10$, dense annotations were provided for semantic scene interpretation, like semantic segmentation and semantic scene completion. For sequences $11-21$, labels for the test set are not provided so that we consider them as test scans. SemanticKITTI contains in total 28 semantic classes, such as vehicles, pedestrians, buildings, roads, etc. Thanks to the effort in [1], the moving-object segmentation benchmark contains three types of classes: 1) unlabeled; 2) moving; and 3) nonmoving. The actually moving vehicles and humans belong to moving objects and all other classes belong to the nonmoving objects. Here, we choose sequence 08 for validation during the training process as [1] does.

For quantitative study in moving-object segmentation performance, we use IoU metric [42] over the objects

$$\text{IoU} = \frac{\text{TP}}{\text{TP}+\text{FP}+\text{FN}} \tag{14}$$

where TP, FP, and FN correspond to the number of true positive, false positive, and false negative predictions for the class.

1) Ablation Study on Input and AR-SI Filter: According to the theory in Section III, the AR-SI filter targets at calculating the rational threshold value to separate moving objects from

TABLE I
IoU Segmentation Result Using Different *P* Value

P value	4	6	8	10	12
ResUnet	0.21	0.25	0.28	0.33	**0.34**
EmPointMovSeg	0.31	0.37	0.42	0.46	**0.47**

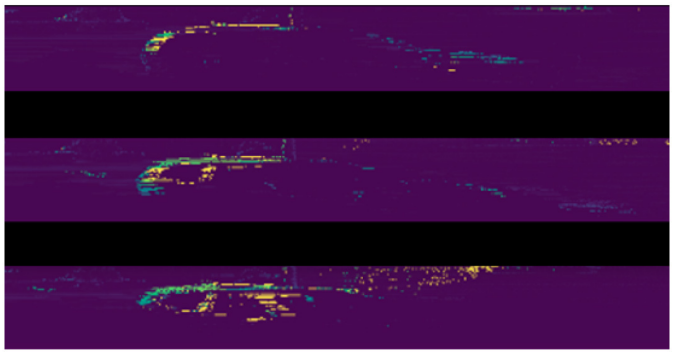

Fig. 6. Residual depth when *P* is set to 4, 8, and 10, respectively. A projected moving car becomes more obvious when the *P* value increases in the residual image.

nonmoving ones. This step involves two modules to generate the final sparse input tensor.

1) Efficient temporal LiDAR sequence module containing rational *P* value in (1), which stands for the number of time steps we need to trackback.
2) Robust AR-SI filter module blocks residual depth error caused by sensor ego-motion noise to affect the sparse convolution network.

For the first module, according to the theory in [15], we compare different *P* values to maximize the IoU performance. Current methods do not give a rational interpretation about the temporal length theoretically. In this article, we leverage the AR-SI theory to calculate *P* value, which presets *P* value to 10 according to [15] and compares the IoU performance around this preset value.

From Table I, we can observe that selecting *P* value to 12 have best IoU performance. While the tracking back length is set to 4, 6, and 8, we can see the IoU is improving because the depth information enhances. This enhancement serves as the temporal feature in sparse tensor so that IoU performance can be improved. However, there is not that much improvement when the *P* value is bigger than 10. The con side of prolonging the temporal sequence is that the residual image requires more time to be ready for the CNN inference (explained in Section IV). We have to balance the computational expense and segmentation accuracy so that 10 is chosen as the most rational sequence length *P* value.

As the AR-SI theory indicates, a static object projected depth equals zero in a sequential residual depth image, which means the same color as the background. In other words, moving ones own contrasting color to the background. As Fig. 6 shows, the residual depth feature of the moving object becomes more obvious when the *P* value increases from 4 to 10. This feature contributes to CNN to get a better IoU performance in the segmentation task. What is more, the residual depth feature of the nonmoving objects

TABLE II
Sparse Convolution Network MOS IoU Results
With/Without AR-SI Filter

	Non-filter	AR-SI filter
ResUnet	0.33	**0.36**
SpSequenceNet	0.42	**0.46**
EmPointMovSeg	0.46	**0.51**

TABLE III
Dense Convolution Network MOS IoU Results
With/Without AR-SI Filter

	Non-filter	AR-SI filter
Unet	0.12	**0.15**
SalsaNext	0.51	**0.53**
KPConv	0.59	**0.61**

becomes zero helps to distinguish these two classes significantly. However, due to the ego-motion sensor error or pose calculation error, nonmoving objects are sometimes considered as moving ones. We use the AR-SI filter to block this error to propagate in the CNN prediction. We prove the AR-SI filter effect in both sparse and dense convolution methods with *P* value fixed in 10. Here, we select or reconstruct ResUnet, SpSequenceNet [43], and our proposed EmPointMovSeg in sparse convolution methods. For dense convolution solutions, Unet [8], SalsaNext [27], and KPConv [44] are chosen, respectively.

We choose SemanticKITTI scan 00 sequence to check all the nonmoving objects. We calculate outlier threshold e_i value via the same theory in [15] which regards a data point outside of (mean $\pm$ 6std) as an outlier. Both Tables II and III indicate the AR-SI filter is able to improve segmentation results in both sparse and dense convolution approaches.

2) Inference Time Study: As shown in Fig. 4, the total inference time contains the residual depth calculation time and the CNN inference time. We fix the *P* value to 10 as the last section discussed and discover the key constraint on time consumption: max dense residual depth calculation uses 251 ms and CNN inference uses 42 ms in scan 00 of SemanticKITTI. Since the LiDAR frequency of SemanticKITTI is around 10 Hz, the traditional method [12], [27] using residual depth is not able to operate the online segmentation task on a powerful server, not even to mention an embedded system.

To overcome this bottleneck, we choose to voxelize each scan of the point cloud to generate a sparse tensor. Less depth information is generated in order to accelerate the residual depth calculation step in inference. The voxel size is the key factor in voxelization, which influences the granularity of a sparse tensor so that efficiency is affected. A smaller voxel size can achieve better IoU performance but efficiency would be reduced. We test the voxel sizes with 0.1, 0.3, 0.5, and 0.7 m in order to discover the best segmentation performance while fulfilling the real-time requirement in LiDAR.

As shown in Table IV, with the AR-SI filter turned on and *P* value fixed to 10, we can get low segmentation performance (0.25) if we want to speed up the EmPointMovSeg using 0.7 m voxel. However, given the practical Velodyne LiDAR

TABLE IV
INFERENCE TIME FOR DIFFERENT VOXEL SIZES

voxel size	0.1m	0.3m	0.5m	0.7m
Time	135.49ms	78.23ms	60.88ms	**52.34ms**
IoU	**0.51**	0.43	0.32	0.25

frequency is below 10 Hz, we choose the voxel size equals 0.3 m to achieve real-time performance. With the model inference time 36.14 ms, the total inference time is 78.23 ms + 36.14 ms = 114.37 ms, which can fulfill the online requirements in LiDAR segmentation with an acceptable IoU performance 43.13%. Comparing to 62.5% [27] in the dense residual image, our EmPointMovSeg suggests a way of balancing computational time and segmentation IoU, which makes online LiDAR moving-object segmentation possible in an embedded system.

3) Moving-Object Segmentation Performance Comparative Study: In this section, we create baselines in the following aspects for comparative study.

1) *No-Seq Nonsparse Method:* This baseline indicates the traditional segmentation method which input raw point cloud data and output segmentation label directly. We do not input sequential data and use each scan raw data to predict the segmentation mask. We choose the most representative method *PointNet* as this baseline. Also, considering the performance on the embedded system, we also add *MINet* [45] into this baseline for comparison. For MINet, the projected image is compulsory input to the network so that the total processing time should take project map calculation into account.
2) *Seq Nonsparse Method:* This baseline means using the sequential raw point cloud containing residual depth information as input. Combing the spacial and temporal information as input, we choose residual depth equals 10 which is the same as our proposed method. We train the network using the label in [1] as the current majority dense segmentation methods. Here, we select the most recent method *SalsaNext* as this candidate. Also, we reconstruct MINet to serve as a sequential segmentation competitor for the reason that: MInet is projected-based multilabel LiDAR segmentation, which is very similar to our proposed method. In order to keep fairness in the comparison, we set the sequence length N to 10 in MINet. The time in total inference progress should add projected image time consumption into the calculation.
3) *No-Seq Sparse Method:* This baseline picks up each frame of raw point cloud converting into a sparse tensor. This sparse tensor, selecting 0.3 m voxel size as discussed in the last section, generates the input of the sparse convolution network which outputs the final prediction result. Here, we use *MinkUNet* in MinkowskiEngine [40], the most representative segmentation method in sparse tensor, to serve as this baseline.
4) *Seq Sparse Method:* The final method, which is our proposed method, *EmPointMovSeg*, use sequential point cloud as input and use the sparse convolution network architecture as Fig. 5 shown. This method works as sequential input and sparse tensor/convolution competitor.

TABLE V
COMPARATIVE STUDY OF DIFFERENT SEGMENTATION METHODS IN IOU AND INFERENCE TIME PERFORMANCE

	IoU	Time
PointNet [16]	0.16	273.78ms
MINet [45]	0.19	**92.21ms**
SalsaNext/N = 10 [27]	**0.62**	293.12ms
MINet/N = 10	0.31	105.29ms
MinkUNet [40]	0.17	106.46ms
EmPointMovSeg	0.43	114.37ms

Since one complete segmentation process involves raw point cloud captured to residual depth images generation if sequential data are used, and inference via CNN network. As a result, residual depth image generation time should be added into the final inference time in case that the temporal sequential data are required. Otherwise, no such time slot is added, and only raw point cloud is used directly. All the residual depth images are calculated in the Python 3.8 environment. The quantitative results for the comparison are shown in Table V.

From Table V, we can see that our proposed EmPointMovSeg is able to achieve online LiDAR data segmentation while maintaining a relatively high IoU performance. Although MINet with sequential input can satisfy the online segmentation requirement, lower IoU is achieved comparing to EmPointMovSeg. Since PointNet baseline uses raw point cloud and process directly, dense and spacial varying density data requires long inference time, which can neither perform online segmentation, nor accurate semantic segmentation in 3-D LiDAR data. MINet is able to achieve the best online performance (21 ms in reference time and 71.21 ms in projected image calculation time). But low IoU performance in movable class segmentation because no sequential feature is imported. SalsaNext serves as our nonsparse convolution competitor using sequence point cloud to calculate residual depth image and then use the CNN network for inference. Although the best achievement in segmentation IoU, we can see dense raw point cloud data generates high residual image computational consumption which is not practical for real-time segmentation. The real LiDAR frequency ranges from 8 to 20 Hz [46], which means online segmentation inference time varying from 125 to 50 ms. Under this real-time constraint, MINet using sequential data can give an acceptable movable class IoU performance. For the rest two methods using a sparse tensor, both MinkUNet and our proposed EmPointMovSeg can satisfy the online segmentation requirements. However, with the poor IoU performance of MinkUNet, our proposed EmPointMovSeg can narrow the gap between accurate segment and real-time performance. Our proposed EmPointMovSeg is not able to achieve the best IoU as dense convolution network SalsaNext which uses sequential and dense input. Instead, considering practical use in the embedded system on an autonomous

TABLE VI
KITTI ODOMETRY BENCHMARK RESULTS

	SuMa++	SuMa+MOS	SuMa+AR-SI
Seq.00-10 (Train)	0.32/0.74	0.32/0.71	**0.31/0.69**
Seq.11-21 (Test)	0.34/1.10	0.34/1.08	**0.32/1.05**

vehicle, we sacrifice IoU for a rational ratio but give the possibility for an online LiDAR segmentation task, which is more important for safety-critical cases.

4) Moving-Object Application Study: In this section, we analyze the effect of AR-SI to assist LiDAR-based odometry/SLAM. SLAM applications highly rely on static objects to converge to precise pose estimation. Dynamic objects should be constructed as outliers in SLAM reprojection error. As a result, MOS and AR-SI methods, both having moving/nonmoving objects separation character, are supposed to improve the rotational accuracy.

Similar to [1], we apply our enhanced prediction mask before feeding the point cloud into the SLAM pipeline in the KITTI odometry benchmark. We evaluate these odometry methods, namely, SuMa++ [47], SuMa+MOS, and our proposed SuMa + AR-SI. We select two key values to illustrate odometry accucacy: 1) relative rotational error in degrees per 100 m and 2) relative translational error in %. The quantitative results in Table VI show MOS and AR-SI methods are the same as our claim of improving the odometry results slightly. Based on the well-designed semantic-enhanced SuMa, our proposed method AR-SI raise up the accuracy a little because the AR-SI method directly targets on filtering out moving objects.

B. Study on Practical Embedded System

In this section, we focus on evaluating our algorithm on a real embedded system. In the aspect of safety, we mainly check two capabilities in detail.

1) Whether the algorithm runtime can fulfill real-time detection requirements.
2) Whether the algorithm can detect a real potential threat to the vehicle.

1) Embedded System Hardware: For the run time hardware as shown in Fig. 7, we choose the NVIDIA Jetson Xavier Developer Kit. This platform contains 512-core NVIDIA Volta GPU with 64 tensor cores, 16-GB 256-bit wide LPDDR4X memory, and 64-bit 8-core NVIDIA Carmel CPU on board. The real-time 3-D LiDAR sensor has 16 channels, a vertical field of view ranging from $+15.0°$ to $-15.0°$, and a 10-Hz rotation rate. The LiDAR sensor communicates with the Xavier board via a gigabit Ethernet interface.

2) Practical Results: The proposed method involves two operations to calculate the final segmentation result: first to use odometry data to generate residual depth, then to combine captured data (x, y, z, remission) with residual depth for model inference.

As shown in Fig. 8, EmPointMovSeg is evaluated in the most representative segmentation dataset, SemanticKITTI. We can reach a consensus that EmPointMovSeg distinguishes

Fig. 7. Embedded system equipped with the Velodyne vlp-16 LiDAR sensor.

the moving car clearly from the static environment around. In Fig. 9, EmPointMovSeg is deployed in the mentioned equipment in Fig. 7 and process real-time point cloud. The original road has a fence and tree on the left side and a truck on the right side. From Fig. 9, we can see clearly our proposed method, EmPointMovSeg, filters out the static objects on the left side and predicts a truck as a moving object using a red mask. Also, benefit from the AR-SI filter, the fence point cloud on the right side is also filtered. This result proves that EmPointMovSeg, not only manages the segmentation task on a resource-limit-embedded system platform but also discover the potential safety-critical threat to the self-driving car. This achievement is meaningful to the practical moving-object segmentation task in autonomous driving.

EmPointMovSeg takes 81.47 ms in pose estimation and input tensor preparation and then uses 36.14 ms for inference of the prediction mask. The total time consumption is 81.47 ms + 36.14 ms = 117.61 ms, which stands for $1/117.61$ ms = 8.5 Hz in the above-mentioned embedded system. This performance can fully satisfy online LiDAR segmentation which is novel to currently existing methods only evaluated in powerful computers or GPU. EmPointMovSeg narrows the gap between accurate segmentation performance and high computation time consumption, which is a vital step for practical LiDAR segmentation use in autonomous driving. What is more, EmPointMovSeg provides a hint

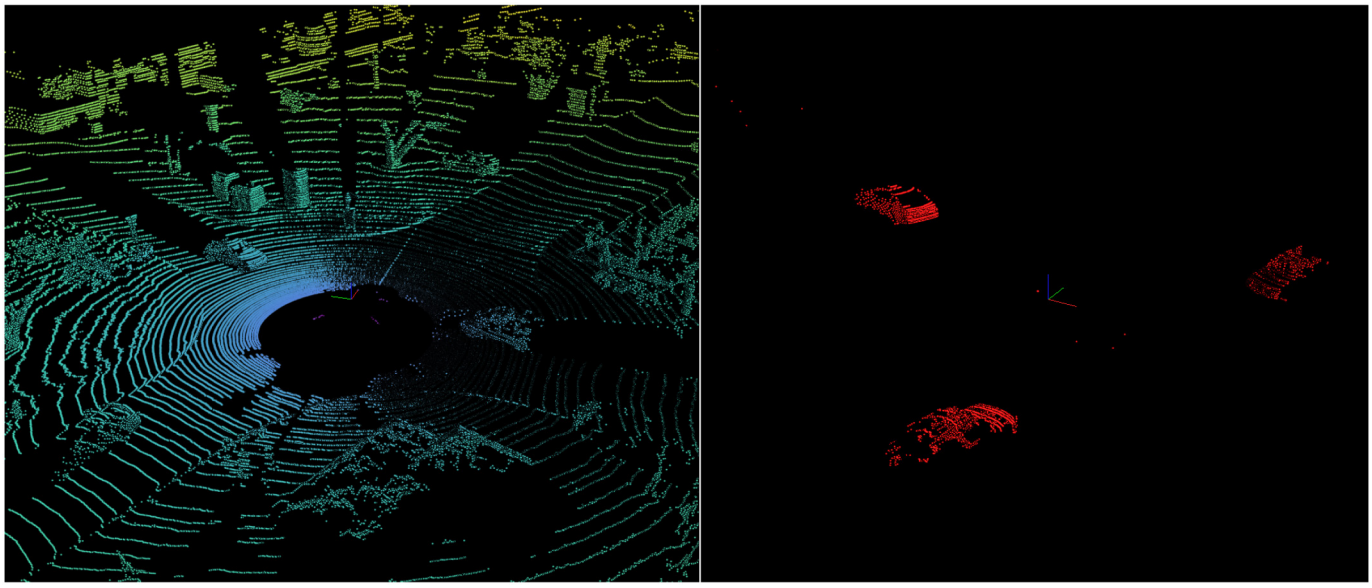

Fig. 8. 3-D LiDAR MOS using EmPointMovSeg in SemanticKITTI.

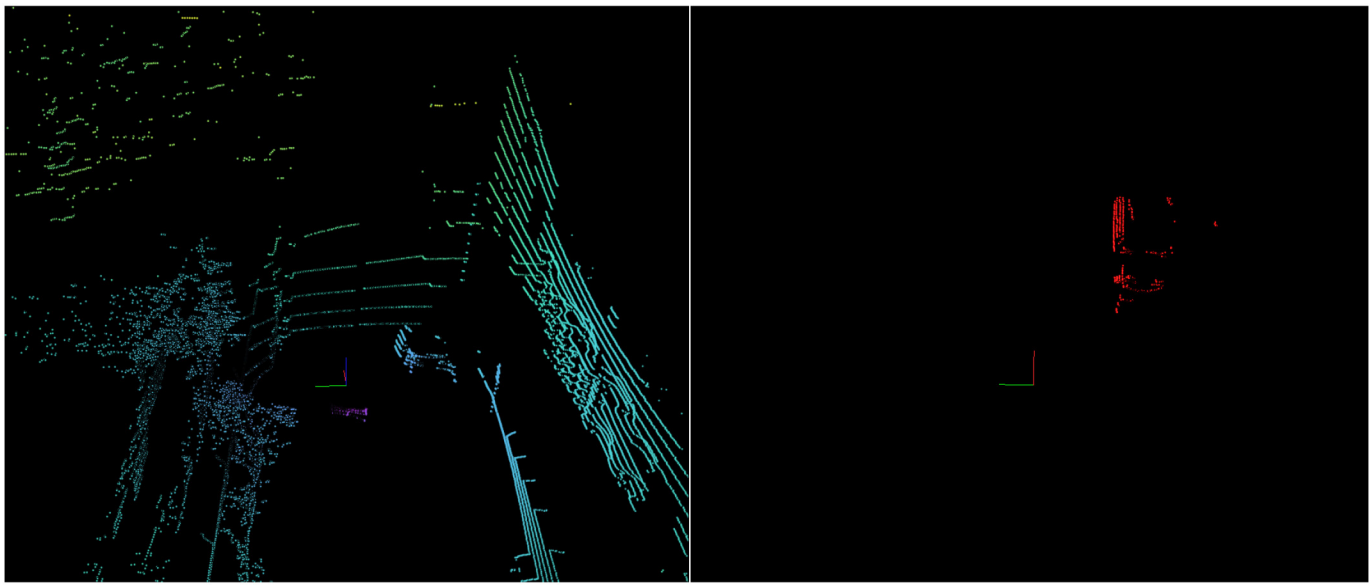

Fig. 9. 3-D LiDAR MOS using EmPointMovSeg in practical HKUST campus.

to accelerate the safety-critical task in autonomous driving reaction rather than optimizing the hardware only. It provides a solution in algorithm calculation that is cost insensitive.

VII. Conclusion

In this work, we first figure out the importance of moving-object segmentation in autonomous driving safety-critical scene context analysis and dig into current drawbacks that constrain moving-object segmentation used in reality. We theoretically solve the moving-object problem using AR-SI math, which is novel compared to current existing large-scale 3-D LiDAR segmentation. Using the advantages of combining the temporal and the spatial features in the AR-SI filter, we propose our sparse tensor-based CNN network, EmPointMovSeg, to balance the process efficiency and segmentation accuracy. The balance is the key in making large-scale 3-D LiDAR segmentation practical in the autonomous driving-embedded system, and most importantly, provides the possibility for the mission of safety-critical task reaction. This algorithm is cost insensitive so that it can be transplanted to other platforms easily. The evaluation results show that our proposed scheme works efficiently in both synthetic dataset and real hardware platforms.

References

[1] X. Chen *et al.*, "Moving object segmentation in 3D LiDAR data: A learning-based approach exploiting sequential data," 2021, *arXiv:2105.08971*.

[2] Z. Shen, L. Han, C. Ma, Z. Jia, T. Li, and Z. Shao, "Leveraging the interplay of raid and SSD for lifetime optimization of flash-based SSD raid," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1395–1408, Jul. 2020.

[3] Y. Lyu, L. Bai, and X. Huang, "Real-time road segmentation using LiDAR data processing on an FPGA," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2018, pp. 1–5.

[4] Y. Li *et al.*, "Deep learning for LiDAR point clouds in autonomous driving: A review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3412–3432, Aug. 2021.

[5] H. Zhou *et al.*, "Cylinder3D: An effective 3D framework for driving-scene LiDAR semantic segmentation," 2020, *arXiv:2008.01550*.

[6] J. Xu, R. Zhang, J. Dou, Y. Zhu, J. Sun, and S. Pu, "RPVNet: A deep and efficient range-point-Voxel fusion network for LiDAR point cloud segmentation," 2021, *arXiv:2103.12978*.

[7] X. Yan *et al.*, "Sparse single sweep LiDAR point cloud segmentation via learning contextual shape priors from scene completion," 2020, *arXiv:2012.03762*.

[8] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput. Assist. Intervent.*, 2015, pp. 234–241.

[9] R. Razani, R. Cheng, E. Taghavi, and L. Bingbing, "Lite-HDSeg: LiDAR semantic segmentation using lite harmonic dense convolutions," 2021, *arXiv:2103.08852*.

[10] M. Gerdzhev, R. Razani, E. Taghavi, and L. Bingbing, "Tornado-Net: Multiview total variation semantic segmentation with diamond inception module," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2021, pp. 9543–9549.

[11] G. Singh, S. Gupta, M. Lease, and C. Dawson, "Range-Net: A high precision streaming SVD for big data applications," 2020, *arXiv:2010.14226*.

[12] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "RangeNet++: Fast and accurate LiDAR semantic segmentation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2019, pp. 4213–4220.

[13] B. Wu, A. Wan, X. Yue, and K. Keutzer, "SqueezeSEG: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3D LiDAR point cloud," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2018, pp. 1887–1893.

[14] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer, "SqueezeSEGV2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, 2019, pp. 4376–4382.

[15] Z. He, Y. Chen, E. Huang, Q. Wang, Y. Pei, and H. Yuan, "A system identification based oracle for control-CPS software fault localization," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, 2019, pp. 116–127.

[16] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 652–660.

[17] Q. Hu *et al.*, "RandLA-Net: Efficient semantic segmentation of large-scale point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11108–11117.

[18] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," 2017, *arXiv:1706.02413*.

[19] vested. "Self-driving technology: LiDAR vs camera." [Online]. Available: https://vested.co.in/blog/self-driving-technology-lidar-vs-camera/

[20] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018, *arXiv:1804.02767*.

[21] Photogrammetry and Robotics Group and the Autonomous Intelligent Systems Group. "SemanticKITTI, a Dataset for semantic scene understanding using LiDAR sequences." [Online]. Available: http://www.semantic-kitti.org/index.html (Accessed: Aug. 25, 2021).

[22] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder–decoder architecture for image segmentation," *IEEE Trans. pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[23] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, 2015, pp. 3431–3440.

[24] J. Xiao, A. Owens, and A. Torralba, "SUN3D: A database of big spaces reconstructed using SFM and object labels," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 1625–1632.

[25] R. Razani, R. Cheng, E. Li, E. Taghavi, Y. Ren, and L. Bingbing, "GP-S3Net: Graph-based panoptic sparse semantic segmentation network," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 16076–16085.

[26] K. Sirohi, R. Mohan, D. Büscher, W. Burgard, and A. Valada, "EfficientLPS: Efficient LiDAR panoptic segmentation," 2021, *arXiv:2102.08009*.

[27] T. Cortinhal, G. Tzelepis, and E. E. Aksoy, "SalsaNext: Fast, uncertainty-aware semantic segmentation of LiDAR point clouds for autonomous driving," 2020, *arXiv:2003.03653*.

[28] R. Cheng, R. Razani, Y. Ren, and L. Bingbing. "S3Net: 3D LiDAR sparse semantic segmentation network." 2021, *arXiv:2103.08745*.

[29] Y. Sun, W. Zuo, H. Huang, P. Cai, and M. Liu, "PointMoSeg: Sparse tensor-based end-to-end moving-obstacle segmentation in 3-D LiDAR point clouds for autonomous driving," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 510–517, Apr. 2021.

[30] T. M. Quan, D. G. Hildebrand, and W.-K. Jeong. "FusionNet: A deep fully residual convolutional neural network for image segmentation in connectomics." 2016. *arXiv:1612.05360*.

[31] J. Gehrung, M. Hebel, M. Arens, and U. Stilla, "An approach to extract moving objects from MLS data using a volumetric background representation," *ISPRS Ann. Photogrammetry Remote Sens. Spatial Inf. Sci.*, vol. 4, May 2017, pp. 107–114.

[32] J. Schauer and A. Nüchter, "The peopleremover—Removing dynamic objects from 3-D point cloud data by traversing a voxel occupancy grid," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1679–1686, Jul. 2018.

[33] P. Ruchti and W. Burgard, "Mapping with dynamic-object probabilities calculated from single 3D range scans," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2018, pp. 6331–6336.

[34] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard, "Rigid scene flow for 3D LiDAR scans," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2016, pp. 1765–1770.

[35] A. Dewan and W. Burgard, "DeepTemporalSeg: Temporally consistent semantic segmentation of 3D LiDAR scans," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2020, pp. 2624–2630.

[36] I. Bogoslavskyi and C. Stachniss, "Efficient online segmentation for sparse 3D laser scans," *Photogrammetry Remote Sens. Geoinf. Sci.*, vol. 85, no. 1, pp. 41–52, 2017.

[37] M. U. Khan, S. Li, Q. Wang, and Z. Shao, "CPS oriented control design for networked surveillance robots with multiple physical constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 5, pp. 778–791, May 2016.

[38] N. Metropolis and S. Ulam, "The Monte Carlo method," *J. Amer. Stat. Assoc.*, vol. 44, no. 247, pp. 335–341, 1949.

[39] F. Zampella, A. R. Jiménez, F. Seco, J. C. Prieto, and J. Guevara, "Simulation of foot-mounted IMU signals for the evaluation of PDR algorithms," in *Proc. Int. Conf. Indoor Position. Indoor Navig. (IPIN)*, Guimaraes, Portugal, Sep. 2011, pp. 1–7.

[40] C. Choy, J. Gwak, and S. Savarese, "4D spatio-temporal ConvNets: Minkowski convolutional neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 3075–3084.

[41] S. Targ, D. Almeida, and K. Lyman, "ResNet in ResNet: Generalizing residual architectures," 2016, *arXiv:1603.08029*.

[42] M. A. Rahman and Y. Wang, "Optimizing intersection-over-union in deep neural networks for image segmentation," in *Proc. Int. Symp. Vis. Comput.*, 2016, pp. 234–244.

[43] H. Shi, G. Lin, H. Wang, T.-Y. Hung, and Z. Wang, "SpsequenceNet: Semantic segmentation network on 4D point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 4574–4583.

[44] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "KPConv: Flexible and deformable convolution for point clouds," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 6411–6420.

[45] S. Li, X. Chen, Y. Liu, D. Dai, C. Stachniss, and J. Gall, "Multi-scale interaction for real-time LiDAR data segmentation on an embedded platform," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 738–745, Apr. 2022.

[46] Velodyne LiDAR. "Velodyne LiDAR puck datasheet." [Online]. Available: https://www.mapix.com/wp-content/uploads/2018/07/63-9229_Rev-H_Puck-_Datasheet_Web-1.pdf

[47] X. Chen, A. Milioto, E. Palazzolo, P. Giguere, J. Behley, and C. Stachniss, "SUMA++: Efficient LiDAR-based semantic SLAM," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2019, pp. 4530–4537.