# LinK: Linear Kernel for LiDAR-based 3D Perception

Tao Lu [1]    Xiang Ding [1]    Haisong Liu [1]    Gangshan Wu [1]    Limin Wang [1,2] *

[1]State Key Laboratory for Novel Software Technology, Nanjing University    [2]Shanghai AI Lab

{taolu,xding,liuhs}@smail.nju.edu.cn, {gswu,lmwang}@nju.edu.cn

## Abstract

*Extending the success of 2D Large Kernel to 3D perception is challenging due to: 1. the cubically-increasing overhead in processing 3D data; 2. the optimization difficulties from data scarcity and sparsity. Previous work has taken the first step to scale up the kernel size from $3 \times 3 \times 3$ to $7 \times 7 \times 7$ by introducing block-shared weights. However, to reduce the feature variations within a block, it only employs modest block size and fails to achieve larger kernels like the $21 \times 21 \times 21$. To address this issue, we propose a new method, called LinK, to achieve a wider-range perception receptive field in a convolution-like manner with two core designs. The first is to replace the static kernel matrix with a linear kernel generator, which adaptively provides weights only for non-empty voxels. The second is to reuse the precomputed aggregation results in the overlapped blocks to reduce computation complexity. The proposed method successfully enables each voxel to perceive context within a range of $21 \times 21 \times 21$. Extensive experiments on two basic perception tasks, 3D object detection and 3D semantic segmentation, demonstrate the effectiveness of our method. Notably, we rank 1st on the public leaderboard of the 3D detection benchmark of nuScenes (LiDAR track), by simply incorporating a LinK-based backbone into the basic detector, CenterPoint. We also boost the strong segmentation baseline's mIoU with **2.7%** in the SemanticKITTI test set. Code is available at https://github.com/MCG-NJU/LinK.*

## 1. Introduction

There is a consensus that a large receptive field contributes positively to many downstream vision tasks. For example, Transformer [2, 3] benefits a lot from the global relation with self-attention and becomes the leading topic in classification [2], segmentation [4], and detection [5]. However, self-attention is not the only route to a large receptive field. Previous works like the RepLKNet [6] and SLaK [7] investigated the potential of obtaining wide-range
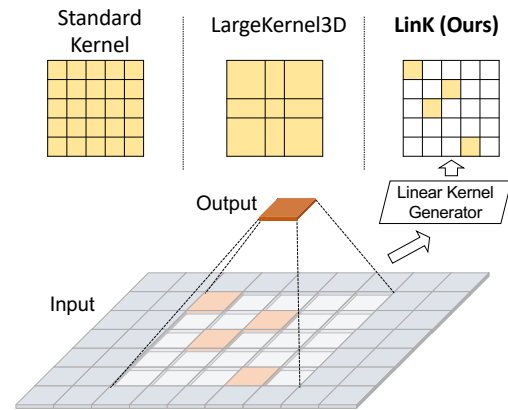
---

*Corresponding author.



Figure 1. Comparisons among the standard kernel, the LargeKernel3D's [1] block-shared kernel, and our LinK's kernel from a generator. Instead of storing a dense kernel matrix, LinK generates sparse kernels online according to the input data. The amount of learnable parameters will not increase along with the kernel size, which enables the scaling up of larger kernel.

information through a large convolutional kernel. They have achieved comparable results with the Transformer-based methods. Considering that the convolution operator is more friendly to existing chip architecture, the large kernel method is efficient in real applications. This raises an immediate question in the 3D perception: can the philosophy of large kernel generalize to the 3D task?

The answer is yes. LargeKernel3D [1] takes the first step and successfully achieves better metrics on both segmentation and detection. Time and space consumption are core concerns during the extension since they increase cubically in the 3D task. LargeKernel3D [1] introduces a spatial sharing kernel to scale the 3D kernel up to $7 \times 7 \times 7$ and restrict the rapid growth of parameters amount. However, compared with the 2D counterparts, which have developed the huge size of $31 \times 31$ [6] and even $51 \times 51$ [7], the $7 \times 7 \times 7$ seems to be not large enough, hence only benefiting from limited context. There are at least two reasons to hinder its size expansion: first, although the parameter amount is under control, the total amounts of operation on each voxel are still increasing cubically; second, its assumption that

the outer parts can share a block-wise weight is too strong to work well in a larger block. So, enlarging the 3D kernel size effectively and efficiently is still a challenging problem.

To handle these issues, we propose a new method, called LinK, to implement a wider-range perception in a convolution-like manner. Two core designs make up the method. The first is to replace the static kernel weights with a linear kernel-generating module to provide weights only for those non-empty areas since the 3D input is very sparse. Meanwhile, this module is layer-wisely shared, which avoids the circumstances that some weights allocated to the blank spaces are not optimized in one iteration. The second is to reuse the pre-computed aggregation results in the overlapped blocks, which makes the computation complexity independent of the kernel size. In other words, we can implement arbitrary kernel sizes with consistent overhead based on the proposed LinK. Brief comparisons among the proposed method and other methods are depicted in Fig 1.

Extensive experiments on the public benchmarks of 3D detection and semantic segmentation tasks demonstrate the effectiveness of LinK. Notably, we achieve the 1st place on the famous 3D detection leaderboard, nuScenes (LiDAR track) [8], by simply replacing the backbone of a classic detection method with the LinK-based backbone. As for the segmentation task, we boost the strong baseline's mIoU with **2.7%** in the SemanticKITTI test split [9]. We will unfold the details in the following sections.

## 2. Related Work

### 2.1. 3D Backbone

According to the input data format (without considering the multi-view 2D representations in this paper), the 3D backbones are grouped into the voxel-based and the point-based method.

Early voxel-based methods [10] directly adopt the 3D convolutional layers to process the volumetric data at the cost of cubic growth of time-space complexity, which forbids a fine voxel resolution. Some researchers propose to optimize the 3D convolution with a compact data structure. Octree-based methods [11, 12] introduce to organize the data into an octree. PVCNN [13] chooses to keep the coarse voxel resolution and compensates for the geometric details with a point branch. Other researchers leverage sparse convolutions to reduce the computation overhead. To solve the submanifold dilation problem of regular sparse convolutions, Graham *et al.* [14] propose submanifold sparse convolutional networks (SSCNs) that keep the same level of sparsity throughout the network. Minkowski-iNet [15] proposes 4-dimensional convolutional neural networks for spatio-temporal perception. All these methods are restricted to small kernel sizes, and they enlarge the

receptive field by stacking more layers. Inspired by RepLKNet [6], LargeKernel3D [1] explores ways to scale up the small kernel to a large one.

To exploit the disorder, point-based methods directly learn from the points' coordinates without any voxelization or projection. The series of PointNet [16, 17] learn point-wise features with MLP and aggregate global features with max-pooling. To imitate the 2D convolution, some position-adaptive kernel generation methods [18–23] are proposed to learn the spatial kernel distribution. Another line of work [24, 25] focus on graph-based networks, which consider each point in a point cloud as a vertex of a graph and generate directed edges based on the neighbors of each point. To purse an efficient network, APP-Net [26] proposes a "push-pull" operator to reduce the redundancy in overhead. Recently, encouraged by the success of Transformer [27] in natural language processing, PCT [28] and Point Transformer [29] design self-attention layers for point clouds and construct Transformer networks for various tasks.

### 2.2. LiDAR-based 3D Perception

**Detection.** LiDAR-based 3D detection aims to predict 3D rotated bounding boxes of objects from point clouds. Here, we mainly focus on outdoor scenarios. SECOND [30] improves VoxelNet [10] by optimizing sparse 3D convolutions. PointPillars [31] replaces voxel representation with a pillar one, which organizes the point clouds in vertical columns for better efficiency. Some researchers also extend 2D detection frameworks to 3D space and achieve remarkable results. Inspired by the R-CNN family [32, 33] for 2D detection, PointRCNN [34] and PV-RCNN [35] use similar detection pipelines with two stages. Center-Point [36] is evolved from CenterNet [37] and Center-Track [38], which adopts a center-based representation for 3D objects. Transfusion-L [39] is a query-based detector (like DETR [5] and Deformable-DETR [40]) and builds a transformer decoder with a small set of object queries.

**Segmentation.** In 3D semantic segmentation, one is required to infer the label of each 3D point. The point-based methods [17, 41] face the overhead issue since their random access to memory is too expensive to deal with the large-scale outdoor scene. Voxel-based methods dominate this task using sparse convolution. Following PVCNN [13], SPVNAS [42] proposes sparse point-voxel convolution. To tackle the imbalanced distribution of points, Cylinder3D [43] proposes a cylindrical partition and builds a 3D cylinder convolution network. RPVNet [44] utilizes three different representations of points (range-based, point-based, and voxel-based) and fuses them into a unified network. For better training of the network, 2DPASS [45] and PVKD [46] enhance the network with distillation strategies.

Unlike these methods focusing on improving the archi-

tectures or the training process, this paper proposes a universal backbone for 3D perception tasks.

## 3. Methodology

This section introduces all of the designs of our method. We begin with two backgrounds to clarify our work's innovation in section 3.1. Then, the detailed procedures are provided in section 3.2. Finally, we show how to incorporate the proposed backbone into two basic 3D perception tasks: object detection and semantic segmentation, in section 3.3.

### 3.1. Background

#### 3.1.1 Introduction to 3D Sparse Convolution

Convolution-based methods aggregate the weighted influences within a pre-specified range. The weights are determined by the local relative positions to the convolutional center. Formula 1 shows the general process of the 3D convolutional operator [14, 15].

$$g_p = \sum_{n \in \mathbb{N}} w_n \cdot f_{p+n}, \qquad (1)$$

where $p$ is the convolution center. $\mathbb{N}$ denotes the non-empty neighbors within a specified range. $f_*$ and $g_*$ are the input and output features, respectively. $w_n$ is the kernel corresponding to the relative location $n$. Different from the 2D images, the Lidar data are spatially sparse. This means that kernels allocated to the empty areas will not participate in the convolution computation, leaving them to fail to be updated during backward propagation. This slows down the optimization process. Meanwhile, regardless of the inputs, the kernel of every location must be stored for the potential calling. This causes a cubically-increasing amount of parameters in a large 3D kernel. For example, for a single convolution layer with a kernel size of $21^3$, $C_{in} = 32$, $C_{out} = 64$, more than 18M learnable parameters are waiting for calling, although most of them will be idle during the inference.

#### 3.1.2 Introduction to the Push-Pull Strategy

The core function of the convolution operation is to introduce spatial interaction. When the convolutional window slides on the feature map, locations covered by the window will be involved in computation with kernels. The overlapped area participates in computation repeatedly, which introduces redundancy. To deal with this issue in the point cloud task, APP-Net [26] proposes a so-called APP operator, which decomposes the spatial interaction into three steps: a push step, $\gamma(p_i \rightarrow proxy)$, to push $p_i$'s feature into a cluster-sharing proxy, an aggregation step in the proxy to fuse cluster-wise information, and a pull step, $\lambda(proxy \rightarrow p_j)$, to pull feature from the auxiliary proxy for

each point $p_j$. Since points within the same cluster share the same aggregation proxy, APP operator avoids the redundant aggregation computation in each cluster. The push step, pull step, and aggregation in the proxy are designed comprehensively to satisfy the requirement that

$$\gamma(p_i \rightarrow proxy) \circ \lambda(proxy \rightarrow p_j) = \eta(p_i \rightarrow p_j). \qquad (2)$$

$\circ$ denotes an operator to combine $\gamma(*)$ and $\lambda(*)$. $\eta(p_i \rightarrow p_j)$ is a function to measure the relation between $p_i$ and $p_j$ directly. According to Formula 2, the influence from the proxy is reducible in the final. The design of APP operator is tightly coupled with the point cloud modality, how to activate its ability in processing voxel data is not explored yet.

### 3.2. Linear Kernel in 3D

According to the statements in section 3.1.1, we conclude that storing the kernel value of every discrete location is neither memory-efficient nor friendly to the optimization process for the 3D large kernel. Thus we propose to adopt a neural network module $w(n)$ to generate the kernel online rather than store the static kernel values $w_n$. This makes the amount of learnable parameters not increase with the kernel size. Furthermore, the empty voxels would not slow down the optimization process.

Although resolving the storing cost and optimization issue, there is still a challenge before adopting the generation module to 3D large kernel. The computations between the kernel and feature map introduce cubically increasing overhead. To deal with it, we provide two critical designs in the following parts: the `Linear Kernel Generator`, and the `Block Based Aggregation`. The whole process is depicted in Fig 2.

#### 3.2.1 Linear Kernel Generator

A larger kernel extracts the input information at the cost of processing each area for more times. We devote to finding reusable common parts in the overlapped area to reduce the overhead.

We start the analysis through a toy example. Given two blocks $A = \{a, b, c, d, e\}$ and $B = \{a, b, c, g, h\}$, where each element denotes a voxel, the overlapped area is $O = \{a, b, c\}$,as shown in Fig 3. We try to aggregate $A$'s features to $a$ and $B$'s features to $b$ using the local offsets, and the influences from the overlapped parts are as follows:

$$\begin{cases} f_{O \rightarrow a} = w(a-a)f_a + w(b-a)f_b + w(c-a)f_c, \\ f_{O \rightarrow b} = w(a-b)f_a + w(b-b)f_b + w(c-b)f_c. \end{cases} \qquad (3)$$

According to Formula 3, each element in $O$ contributes to $a$ and $b$ using different offsets, so we cannot reuse the over-
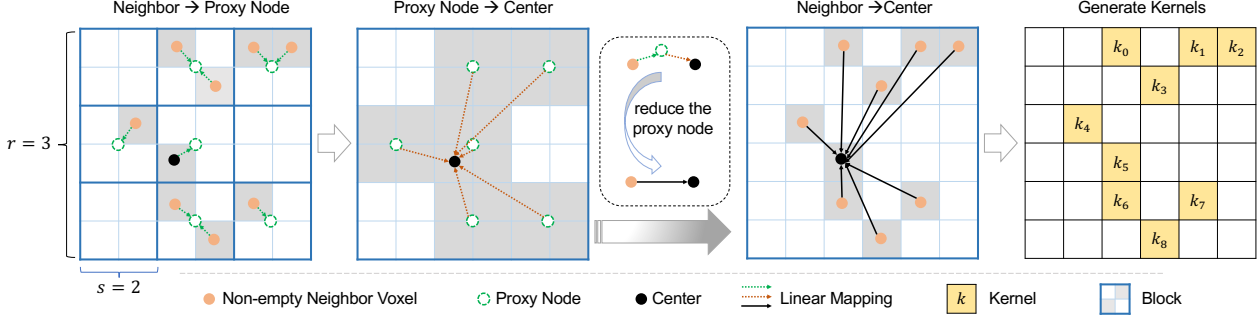
Figure 2. The procedure of LinK. The input is partitioned into non-overlapped blocks (the block size is $s^3$). Each non-empty voxel pushes its feature to a block-wise proxy, and then the center voxel pulls features only from the neighbor proxies (neighbor range is $r^3$). The push and pull processes are driven in a reducible manner (details in 3.2.2) such that the block-wise proxy supports reusing for every potential calling. The resulting matrix serves as a convolutional kernel to weigh neighbors.
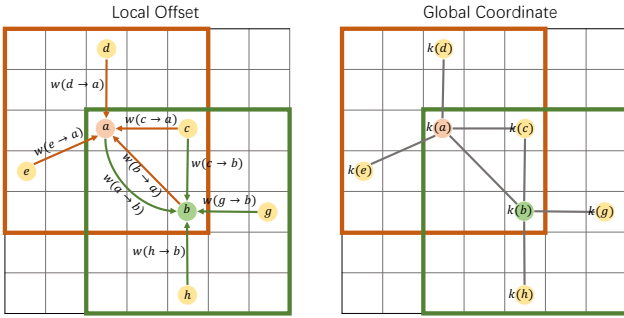


Figure 3. The comparison between modeling the local offsets and global coordinates. $w(x-y)$ actively measures how $x$ influences $y$ through modeling the offset from $x$ to $y$. $k(x)$ generates kernel for $x$ using its global coordinates. When $y$ queries neighbor features, the relation between $x$ and $y$ can be composed from $k(x)$ and $k(y)$.

lapped aggregation results through modeling the local offsets.

Considering that the global coordinate for each voxel is fixed and unique, we think about decomposing the local offset into combinations of the global coordinates. Specifically, we first define a new kernel generator as follows:

$$k(x) = \Psi(\sigma(x)) \qquad (4)$$

where $\sigma(x) = W \times x$ is a linear mapping function, $W \in \mathbb{R}^{C_{in} \times 3}$. $\Psi(*)$ is an activation function. Inspired by APP-Net [26], we adopt trigonometric functions for the activation, e.g.,

$$\{k^{(0)}(x) = \cos(\sigma(x)), \quad k^{(1)}(x) = \sin(\sigma(x))\}, \quad (5)$$

which provides the following relation based on the sum-to-product formula:

$$\begin{aligned} k^{(0)}(x - y) =& \cos(\sigma(x)) \cdot \cos(\sigma(y)) \\ &+ \sin(\sigma(x)) \cdot \sin(\sigma(y)) \\ =& k^{(0)}(x) \cdot k^{(0)}(y) + k^{(1)}(x) \cdot k^{(1)}(y). \quad (6) \end{aligned}$$

where $x$ and $y$ are the global coordinates, and $x - y$ is local offset. Equation 6 decomposes local offset into global positions.

Then we compute the following two auxiliary aggregations for the overlap area:

$$\begin{cases} f_O^{(0)} = k^{(0)}(a) \cdot f_a + k^{(0)}(b) \cdot f_b + k^{(0)}(c) \cdot f_c, \\ f_O^{(1)} = k^{(1)}(a) \cdot f_a + k^{(1)}(b) \cdot f_b + k^{(1)}(c) \cdot f_c. \end{cases} \quad (7)$$

$f_O^{(0)}$ and $f_O^{(1)}$ are reusable between $A$ and $B$ since they are computed based on the fixed global locations. To obtain the final aggregations of the overlapped area for the center voxel $a$, we leverage the auxiliary aggregations in the following manner:

$$\begin{aligned} f_{O \to a} =& f_O^{(0)} \cdot k^{(0)}(a) + f_O^{(1)} \cdot k^{(1)}(a) \\ =& \sum_{p \in O} k^{(0)}(p - a) \cdot f_{a+(p-a)}. \end{aligned} \quad (8)$$

Combined Formula 7 with Formula 6, the $f_{O \to a}$ models the local offsets. And the Formula 8 is an instantiation of the convolutional operator in Formula 1. We call this process `Linear Kernel Generator` to emphasize the linear mapping in the core part.

### 3.2.2 Block Based Aggregation

The above linear kernel enables reusing the overlapped area. Then a new question emerges: how to set up the overlap area? Motivated by ViT [2], we partition the entire input space into several non-overlapped blocks. Specifically, for the input scene $P \in \mathbb{Z}^{N \times 3}$, we quantize each voxel $p$'s coordinate with a block size $s$ and compute the corresponding hash code $l$ as follows:

$$l = Hash(\lfloor \frac{p(0)}{s} \rfloor, \lfloor \frac{p(1)}{s} \rfloor, \lfloor \frac{p(2)}{s} \rfloor). \quad (9)$$

(a) Backbone.



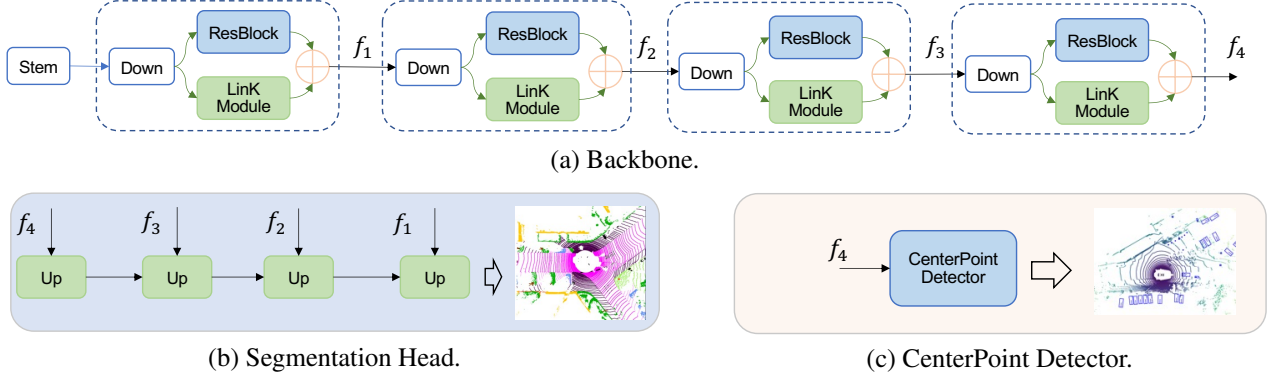(b) Segmentation Head.



(c) CenterPoint Detector.

Figure 4. (a) Architecture of the LinK-based backbone; (b) the constructed network for 3D semantic segmentation; (c) the constructed network for 3D object detection.

Voxels owning the same hash code will be grouped into the same block. The block collection is denoted as $\mathbf{B} = \{B_0, B_1, ..., B_m\}$. Based on Formula 7, we conduct block-wise proxy aggregation for reusing as follows:

$$\begin{cases} f_{B_i}^{(0)} = \sum_{x \in B_i} k^{(0)}(x) \cdot f_x, \\ f_{B_i}^{(1)} = \sum_{x \in B_i} k^{(1)}(x) \cdot f_x. \end{cases} \quad (10)$$

The $f_{B_i}^{(0)}$ and $f_{B_i}^{(1)}$ carry information with a receptive field of $s^3$.

To expand the receptive field, we query the neighboring blocks' aggregation results for each block with a query range of $r^3$. For block $B_i$, denoting its neighboring block set as $\mathbb{B}_i$, then we compute the expanded block aggregation as

$$\begin{cases} f_{\mathbb{B}_i}^{(0)} = \sum_{j \in \mathbb{B}_i} f_{B_j}^{(0)}, \\ f_{\mathbb{B}_i}^{(1)} = \sum_{j \in \mathbb{B}_i} f_{B_j}^{(1)}. \end{cases} \quad (11)$$

The $g_{\mathbb{B}_i}^{(0)}$ and $g_{\mathbb{B}_i}^{(1)}$ carry information with a receptive field of $(r \times s)^3$. For voxel $x$ within the block $B_i$, its final feature is updated by

$$g_x = \frac{1}{N_{\mathbb{B}_i}} [g_{\mathbb{B}_i}^{(0)} \cdot k^{(0)}(x) + g_{\mathbb{B}_i}^{(1)} \cdot k^{(1)}(x)] \quad (12)$$

$N_{\mathbb{B}_i}$ is the count of non-empty voxels. The voxel $x$ aggregates the information within a $(r \times s)^3$ area in a convolution-like manner.

### 3.2.3 Enhancements to the Kernel Generation

To enhance the representation, two simple strategies from different views are proposed to improve kernel generation.
**Learnable Frequency for the Activation** For enhancing the model capability, we introduce two improvements to the cos-based and sin-based activation: channel-wise learnable parameters $\alpha$ to adjust the frequency, and an identity term $+x$ to preserve spatial information. The augmented activation function $\Psi'(*)$ is as follows:
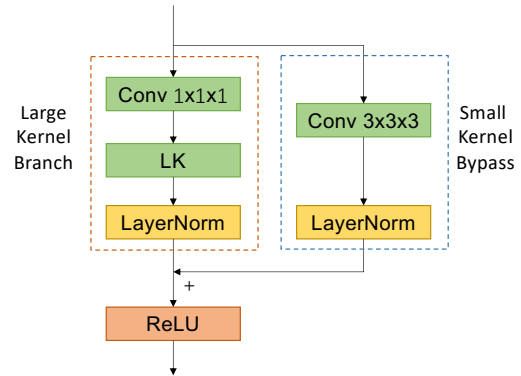


Figure 5. Structures of LinK Module. The LK branch is responsible for a large kernel size while the Conv$3 \times 3 \times 3$ bypass makes up for the delicate local structures.

$$\Psi'(x) = \Psi(\alpha \cdot x) + x \quad (13)$$

**Group Sharing Weight** The effective receptive field is significantly enlarged by block-based aggregation. Since each voxel only contributes once to the kernel generation, a large kernel range makes learning the kernel of each offset difficult. To facilitate the optimization, we adopt a group-sharing policy. Specifically, for the $C_{in}$ input channels, we only generate kernels of $\frac{C_{in}}{\#groups}$ channels and let every $\#groups$ channels share the same weight. Thus every weight would have more chances to be updated. We use $\#groups = 2$ in practice.

### 3.3. Network Structure

#### 3.3.1 LinK Module

Since LinK aggregates spatial information in a depth-wise manner, we apply a $1^3$ convolution to the input feature before sending it to the LinK operator to introduce channel mixing [53, 54]. Meanwhile, following the previous practice [1, 6], we append a parallel $3^3$ convolutional branch to preserve detailed structures. This operation also stabilizes

Table 1. Results on the test phase of nuScenes Detection. **Bold**: best results. * denotes using TTA.

| Methods | Source | NDS | mAP | car | truck | bus | trailer | construction _vehicle | pedestrian | motorcycle | bicycle | traffic_cone | barrier |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PointPillars [31] | *CVPR19* | 45.3 | 30.5 | 68.4 | 23.0 | 28.2 | 23.4 | 4.1 | 59.7 | 27.4 | 1.1 | 30.8 | 38.9 |
| 3DSSD [47] | *CVPR20* | 56.4 | 42.6 | 81.2 | 47.2 | 61.4 | 30.5 | 12.6 | 70.2 | 36.0 | 8.6 | 31.1 | 47.9 |
| CenterPoint [36] | *CVPR21* | 65.5 | 58.0 | 84.6 | 51.0 | 60.2 | 53.2 | 17.5 | 83.4 | 53.7 | 28.7 | 76.7 | 70.9 |
| HotSpotNet [48] | *ECCV20* | 66.0 | 59.3 | 83.1 | 50.9 | 56.4 | 53.3 | 23.0 | 81.3 | 63.5 | 36.6 | 73.0 | 71.6 |
| TransFusion-L [39] | *CVPR22* | 70.2 | 65.5 | 86.2 | **56.7** | 66.3 | 58.8 | 28.2 | 86.1 | 68.3 | 44.2 | **82.0** | **78.2** |
| Focals Conv [49] | *CVPR22* | 70.0 | 63.8 | **86.7** | 56.3 | **67.7** | 59.5 | 23.8 | **87.5** | 64.5 | 36.3 | 81.4 | 74.1 |
| LargeKernel [1] | *arXiv22* | 70.5 | 65.3 | 85.9 | 55.3 | 66.2 | 60.2 | 26.8 | 85.6 | 72.5 | 46.6 | 80.0 | 74.3 |
| LinK | *Ours* | **71.0** | **66.3** | 86.1 | 55.7 | 65.7 | **62.1** | **30.9** | 85.8 | **73.5** | **47.5** | 80.4 | 75.5 |
| VISTA* [50] | *CVPR22* | 70.4 | 63.7 | 84.7 | 54.2 | 64.0 | 55.0 | 29.1 | 83.6 | 71.0 | 45.2 | 78.6 | 71.8 |
| UVTR-LiDAR* [51] | *NeurIPS22* | 69.7 | 63.9 | 86.3 | 52.2 | 62.8 | 59.7 | 33.7 | 84.5 | 68.8 | 41.1 | 74.7 | 74.9 |
| MDRNet* [52] | *arXiv22* | 72.8 | 68.4 | **87.9** | 58.5 | 67.3 | 64.1 | 30.2 | **89.0** | 77.0 | 50.7 | **85.0** | 74.7 |
| LargeKernel3D* [1] | *arXiv22* | 72.8 | 68.8 | 87.3 | 59.1 | 68.5 | 65.6 | 30.2 | 88.3 | 77.8 | 53.5 | 82.4 | 75.0 |
| LinK* | *Ours* | **73.4** | **69.8** | 87.3 | **60.2** | **69.8** | 65.9 | **34.0** | 88.2 | **78.8** | 54.3 | 83.0 | **76.8** |

the optimization process. The resulting architecture is illustrated in Fig 5. Unlike the choices in LargeKernel3D [1], we do not adopt a dilation $> 1$ for the $3^3$ branch. And we replace the BatchNormalization [55] with LayerNormalization [56] to reinforce those informative channels.

### 3.3.2 Applications in Perception Tasks

LinK is incorporated into two essential perception tasks: 3D object detection and 3D semantic segmentation. We choose two representative architectures for the two tasks and directly replace their SparseConv-based backbone with the LinK-based backbone and keep the original design of their segmentation head and detector. Detailed architectures are shown in Fig 4.

## 4. Experiments

To verify the effectiveness of our method and explore its characteristics, we conduct extensive experiments in this section. The whole project is implemented upon three software architectures: PyTorch [57], TorchSparse [58], and SpConv [59]. All experiments are conducted on a server with 4 RTX 3090 GPUs. We present two most representative datasets in this section. Results on more datasets can be found in the supplementary material.

### 4.1. 3D Object Detection

#### 4.1.1 Dataset

We evaluate the 3D detection performance on the widely used benchmark, nuScenes [8], a public dataset for autonomous driving. It is collected from challenging urban scenes, consisting of 1000 annotated sequences. Among them, 700 sequences are used as the training phase, 150 as validation, and the remaining 150 as the test phase for online evaluation. The bounding boxes are labeled with not
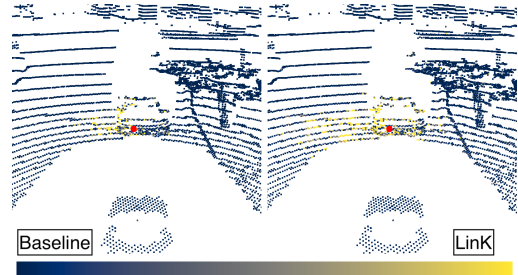


Figure 6. The effective receptive field (ERF) of the detection. The brightness indicates the degree of activation. LinK enjoys a wider-range perception.

only categories but also some attributes like the velocity, scale, orientation and translation. Besides the point cloud data from LiDAR, it also provides the $360°$ image modality from cameras and signal from the Radar sensor. In this paper, we only utilize the point cloud from LiDAR. The evaluation metrics include the mAP and a dataset-related NDS, i.e., nuScenes Detection Score. The NDS measures the comprehensive performance by combining the mAP and other attributes in a weighted manner.

#### 4.1.2 Implementation Details

The kernel configuration for the detection is $\{r = 3, s = 7\}$. We implement the detection codebase by replacing the original backbone of CenterPoint [36] with a LinK-based backbone. For a fair comparison, we keep all the original hyperparameters in CenterPoint [36] to train our network. Following the common practice, the training phase is augmented with the CBGS [65] and gt-sampling [30] strategy to balance the long-tail issues. During test and validation, we follow LargeKernel3D [1] to report the plain inference result and the test-time augmented (TTA, including flip and rotation) inference results.

Table 2. SemanticKITTI test results. Red: surpassing the baseline; **bold**: best results; 'P': point cloud; 'R': range map; 'V': voxel.

| Method | Input | mIoU | Car | Bicycle | Motorcycle | Truck | Other-vehicle | Person | Bicyclist | Motorcyclist | Road | Parking | Sidewalk | Other-ground | Building | Fence | Vegetation | Trunk | Terrain | Pole | Traffic-sign |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RandLA-Net [41] | P | 53.9 | 94.2 | 26.0 | 25.8 | 40.1 | 38.9 | 49.2 | 48.2 | 7.2 | 90.7 | 60.3 | 73.7 | 20.4 | 86.9 | 56.3 | 81.4 | 61.3 | 66.8 | 49.2 | 47.7 |
| RangeNet++ [60] | R | 52.2 | 91.4 | 25.7 | 34.4 | 25.7 | 23.0 | 38.3 | 38.8 | 4.8 | 91.8 | 65.0 | 75.2 | 27.8 | 87.4 | 58.6 | 80.5 | 55.1 | 64.6 | 47.9 | 55.9 |
| SqueezeSegV3 [61] | R | 55.9 | 92.5 | 38.7 | 36.5 | 29.6 | 33.0 | 45.6 | 46.2 | 20.1 | 91.7 | 63.4 | 74.8 | 26.4 | 89.0 | 59.4 | 82.0 | 58.7 | 65.4 | 49.6 | 58.9 |
| SalsaNext [62] | R | 59.5 | 91.9 | 48.3 | 38.6 | 38.9 | 31.9 | 60.2 | 59.0 | 19.4 | 91.7 | 63.7 | 75.8 | 29.1 | 90.2 | 64.2 | 81.8 | 63.6 | 66.5 | 54.3 | 62.1 |
| SPVNAS [42] | P+V | 67.0 | 97.2 | 50.6 | 50.4 | 56.6 | 58.0 | 67.4 | 67.1 | 50.3 | 90.2 | 67.6 | 75.4 | 21.8 | 91.6 | 66.9 | 86.1 | 73.4 | 71.0 | 64.3 | 67.3 |
| Cylinder3D [43] | V | 67.8 | 97.1 | 67.6 | 64.0 | 59.0 | 58.6 | 73.9 | 67.9 | 36.0 | 91.4 | 65.1 | 75.5 | 32.3 | 91.0 | 66.5 | 85.4 | 71.8 | 68.5 | 62.6 | 65.6 |
| (AF)2-S3Net [63] | V | 69.7 | 94.5 | 65.4 | **86.8** | 39.2 | 41.1 | **80.7** | **80.4** | **74.3** | 91.3 | 68.8 | 72.5 | **53.5** | 87.9 | 63.2 | 70.2 | 68.5 | 53.7 | 61.5 | **71.0** |
| DRINet [64] | P+V | 67.5 | 96.9 | 57.0 | 56.0 | 43.3 | 54.5 | 69.4 | 75.1 | 58.9 | 90.7 | 65.0 | 75.2 | 26.2 | 91.5 | 67.3 | 85.2 | 72.6 | 68.8 | 63.5 | 66.0 |
| RPVNet [44] | R+P+V | 70.3 | **97.6** | **68.4** | 68.7 | 44.2 | 61.1 | 75.9 | 74.4 | 73.4 | **93.4** | **70.3** | **80.7** | 33.3 | **93.5** | **72.1** | **86.5** | **75.1** | **71.7** | **64.8** | 61.4 |
| Mink(baseline) [15] | V | 68.0 | 97.1 | 51.8 | 56.4 | 43.3 | 56.8 | 70.2 | 75.7 | 51.8 | 89.9 | 67.8 | 74.8 | 32.9 | 91.5 | 66.5 | 86.2 | 74.6 | 71.0 | 63.5 | 70.0 |
| LinK(Ours) | V | **70.7** | 97.4 | 58.4 | 56.6 | 52.9 | **64.2** | 72.3 | 77.0 | 69.1 | 90.6 | 68.2 | 76.2 | 34.5 | 92.0 | 68.8 | 85.7 | 74.3 | 70.5 | **64.8** | 69.5 |

Table 3. Val@nuScenes Det.

| Methods | NDS | mAP |
|---|---|---|
| CBGS [65] | 62.6 | 51.4 |
| CenterPoint [36] | 66.4 | 59.0 |
| HotSpotNet [48] | 66.0 | 59.5 |
| TransFusion-L [39] | 66.8 | 60.0 |
| Focals Conv [49] | 68.1 | 61.2 |
| LargeKernel3D [1] | 69.1 | 63.3 |
| LinK | **69.5** | **63.6** |

Table 4. Val@SemKITTI Seg.

| Methods | mIoU |
|---|---|
| RandLA-Net [41] | 57.1 |
| RangeNet++ [60] | 57.3 |
| SPVNAS [42] | 64.7 |
| Cylinder3D [43] | 63.8 |
| RPVNet [44] | 65.5 |
| Mink [15] | 66.1 |
| LinK | **67.5** |

### 4.1.3 Results

We compare the detection results with many representative methods. All the test results in Table 1 are obtained through public sources, like published papers or competition websites. Until the time of paper submission, our method ranks 1st on the public LiDAR detection leaderboard. Especially for the NDS metric, which has been stuck in 72.7%∼ 72.8% for more than one year, we are the first method to obtain a result higher than 73% on the LiDAR track, which confirms the superiority of our large kernel method. Table 3 shows the comparisons on the validation phase. We achieve consistent improvement in both settings. The effective receptive field is shown in Fig 6, and our approach enjoys a larger ERF than the baseline method. More qualitative results are available in the supplementary material.

## 4.2. 3D Semantic Segmentation

### 4.2.1 Dataset

We evaluate the semantic segmentation performance on the SemanticKITTI [9]. It is a large-scale (including more than 43000 scans, and each scan has more than 100k points) autonomous dataset constructed by labeling the odometry dataset KITTI [66] with 20 categories of semantic masks. The dataset contains 22 sequences and has been officially partitioned into three phases: sequences [00-07, 09, 10] as the training phase, sequence 08 as the validation phase, and the rest sequences [11-21] as the online testing benchmark. The evaluation metric is the mean Intersection over Union.

### 4.2.2 Implementation Details

The kernel configuration for the segmentation is $\{r = 2, s = 3\}$. We train the network for 25 epochs in total. Following SPVNAS [42], we adopt an initial learning rate of 2.4e-1 and adjust it using the cosine scheduler. The optimizer for updating learnable parameters is SGD. All experiments are run with 4 GPUs in parallel, and the batch size in each GPU is 2. To deal with different object scales, we follow previous works [43, 44] to introduce the Lovasz loss [67] to cooperate with the original cross-entropy loss. The size for voxelization is 0.05m. We preserve 80,000 points for each scan to train the network. During the validation and test processes, we report the result of direct inference and the TTA results. Due to the severe long-tail problem, this dataset is very sensitive to some small categories, like the bicycle, person, and motorcyclist. Some model-ensemble techniques and instance-level augmentations (like conducting the Copy-Paste and Cut-Mix on some small categories) contribute significantly to the final results. Because there are no comprehensive works to conclude these tricks in a unified manner, we do not adopt these augmentations to prevent from shadowing the essence of model design.

### 4.2.3 Results

The LinK for segmentation is implemented by only replacing the Mink's [15] encoder with a LinK-based encoder. So, the Mink serves as a direct baseline. Table 2 shows the test results on SemanticKITTI [9]. Most baseline results are obtained from the public sources. For Mink [15], we reproduce it with the same configurations as ours and surprisingly find that this basic architecture hits performance on par with other complex architectures. When combined with the LinK module to enlarge the receptive field, we achieve an improvement of 2.7% in mIoU. Table 4 shows the results of the validation phase. Visulaizations are provided in the supplementary material.

Table 5. Performance on different scale objects.

| Category | Size($m^3$) | Detection | | Segmentation | |
|---|---|---|---|---|---|
| | | Center Point | +LinK | Mink | +LinK |
| Truck | $6 \times 2 \times 2$ | 51.0 | (+4.7)55.7 | 43.3 | (+9.6)52.9 |
| Person | $0.4 \times 0.4 \times 2$ | 83.4 | (+2.4)85.8 | 70.2 | (+2.1)72.3 |

Table 6. Time and parameters analysis for semantic segmentation.

| Methods | Parameters | Runtime(ms) | mIoU(%) |
|---|---|---|---|
| Mink | 8.5M | 69 | 66.1 |
| Conv$7 \times 7 \times 7$ | 21.75M | 139 | 66.8 |
| LinK | 10.75M | 87 | 67.5 |

Table 7. Time and parameters analysis for 3D object detection.

| Methods | Parameters | Runtime(ms) | mAP | NDS |
|---|---|---|---|---|
| CenterPoint | 8.6M | 73 | 59.0 | 66.4 |
| LinK | 10.3M | 109 | 60.3 | 67.7 |

## 4.3. Overhead Analysis

We measure the number of parameters and the inference speed to evaluate LinK's practicability in Table 6 and Table 7. As shown in Table 6, we achieve a better mIoU by consuming fewer computation resources than the naive Conv$7 \times 7 \times 7$ operator in the segmentation task. We measure the performance on a single 3090 GPU with bs=1.

## 4.4. Ablation Studies

**How large kernel play a role in 3D perception?** We investigate this question through two ways. First, according to Fig 6, LinK-based backbone produces a wider range of ERF. Second, as shown in Table 5, big objects benefit more from the large kernel. In conclusion, we think the large kernel enhances the network's ability to model different scales objects more effectively with fewer layers.

**The branch combinations in LinK module.** We ablate the construction of the LinK-based architecture from two aspects: (1). does the large kernel really contribute positively? (2). its comparisons with the standard large kernel (Conv$7 \times 7 \times 7$). According to Table 8, both the standard kernel of $7 \times 7 \times 7$ and the LinK improve the baseline with the aid of a $3 \times 3 \times 3$ bypass branch, which verifies the effectiveness of a large kernel. Meanwhile, the LinK outperforms the standard large kernel with a non-trivial margin. When removing the ResBlock branch, i.e., the backbone only consists of large kernels, the network still hits a high mAcc. This implies the large kernel's ability in modeling large-scale objects.

**The kernel size.** We explore the influence of LinK's kernel size in the detection task. As shown in Fig 7, the performances increase along with the kernel size within a range and saturate when the kernel size is larger than $(3 \times 7)^3$.

**The two strategies to augment kernel.** We validate the effectiveness of the two augmentations to the kernel generation. Table 9 shows that introducing the learnable frequency works well with a relatively small kernel, and the grouping
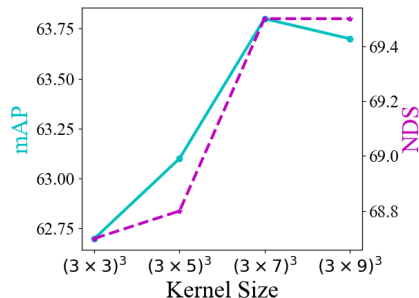


Figure 7. Detection performance with different kernel sizes.

Table 8. Ablations on the backbone structures. 'RB': ResBlock, 'LKB': large kernel branch, 'LK': how to implement large kernel.

| RB | LKB | | mIoU(%)↑ | mAcc(%)↑ |
|---|---|---|---|---|
| | Bypass | LK | | |
| | | | 66.1 | 72.4 |
| | ✓ | | 66.4 | 72.9 |
| | ✓ | Conv$7 \times 7 \times 7$ | 66.8 | 72.3 |
| ✓ | | Conv$7 \times 7 \times 7$ | 66.1 | 72.3 |
| | | LinK | 65.6 | 72.3 |
| | ✓ | LinK | **67.5** | **74.7** |
| | ✓ | LinK | 65.7 | 73.6 |

Table 9. Ablations on the two augmentations of kernel.

| $r \times s$ | Learnable Frequency | Group Sharing | mIoU(%) |
|---|---|---|---|
| | | | 67.3 |
| $2 \times 3$ | ✓ | | **67.5** |
| | | ✓ | 67.1 |
| | | | 66.2 |
| $3 \times 5$ | ✓ | | 66.8 |
| | | ✓ | **67.5** |

sharing weight performs better with a larger kernel.

## 5. Conclusion

A large receptive field is essential in computer vision tasks. In this paper, we have posed a linear kernel generator, LinK, to enlarge the effective receptive field for 3D perception tasks at the cost of moderate computations. Extensive experimental results on the detection and segmentation demonstrate the effectiveness of the proposed LinK, and we achieve consistent improvements over baselines. In the future, we will work on generalizing this method to more basic models like the Transformer and dynamic convolution.

# References

[1] Yukang Chen, Jianhui Liu, Xiaojuan Qi, Xiangyu Zhang, Jian Sun, and Jiaya Jia. Scaling up kernels in 3d cnns. *arXiv preprint arXiv:2206.10555*, 2022. 1, 2, 5, 6, 7

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1, 4

[3] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 1

[4] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021. 1

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 1, 2

[6] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11963–11975, 2022. 1, 2, 5

[7] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Mykola Pechenizkiy, Decebal Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022. 1

[8] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 2, 6

[9] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019. 2, 7

[10] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 922–928. IEEE, 2015. 2

[11] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3577–3586, 2017. 2

[12] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions On Graphics (TOG)*, 36(4):1–11, 2017. 2

[13] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[14] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9224–9232, 2018. 2, 3

[15] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 2, 3, 7

[16] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 2

[17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 2

[18] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019. 2

[19] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019. 2

[20] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019. 2

[21] Alexandre Boulch. Convpoint: Continuous convolutions for point cloud processing. *Computers & Graphics*, 88:24–34, 2020. 2

[22] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018. 2

[23] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018. 2

[24] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on

graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017. 2

[25] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 2

[26] Tao Lu, Chunxu Liu, Youxin Chen, Gangshan Wu, and Limin Wang. App-net: Auxiliary-point-based push and pull operations for efficient point cloud classification. *arXiv preprint arXiv:2205.00847*, 2022. 2, 3, 4

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2

[28] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7(2):187–199, 2021. 2

[29] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16259–16268, 2021. 2

[30] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 2, 6

[31] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019. 2, 6

[32] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 2

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 2

[34] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 770–779, 2019. 2

[35] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020. 2

[36] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11784–11793, 2021. 2, 6, 7

[37] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for

object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6569–6578, 2019. 2

[38] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points. In *European Conference on Computer Vision*, pages 474–490. Springer, 2020. 2

[39] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1090–1099, 2022. 2, 6, 7

[40] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020. 2

[41] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020. 2, 7

[42] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *European conference on computer vision*, pages 685–702. Springer, 2020. 2, 7

[43] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. *arXiv preprint arXiv:2011.10033*, 2020. 2, 7

[44] Jianyun Xu, Ruixiang Zhang, Jian Dou, Yushi Zhu, Jie Sun, and Shiliang Pu. Rpvnet: A deep and efficient range-point-voxel fusion network for lidar point cloud segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16024–16033, 2021. 2, 7

[45] Xu Yan, Jiantao Gao, Chaoda Zheng, Chao Zheng, Ruimao Zhang, Shuguang Cui, and Zhen Li. 2dpass: 2d priors assisted semantic segmentation on lidar point clouds. In *European Conference on Computer Vision*, pages 677–695. Springer, 2022. 2

[46] Yuenan Hou, Xinge Zhu, Yuexin Ma, Chen Change Loy, and Yikang Li. Point-to-voxel knowledge distillation for lidar semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 8479–8488, 2022. 2

[47] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3dssd: Point-based 3d single stage object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11040–11048, 2020. 6

[48] Qi Chen, Lin Sun, Zhixin Wang, Kui Jia, and Alan Yuille. Object as hotspots: An anchor-free 3d object detection approach via firing of hotspots. In *European conference on computer vision*, pages 68–84. Springer, 2020. 6, 7

[49] Yukang Chen, Yanwei Li, Xiangyu Zhang, Jian Sun, and Jiaya Jia. Focal sparse convolutional networks for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5428–5437, 2022. 6, 7

[50] Shengheng Deng, Zhihao Liang, Lin Sun, and Kui Jia. Vista: Boosting 3d object detection via dual cross-view spatial attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8448–8457, 2022. 6

[51] Yanwei Li, Yilun Chen, Xiaojuan Qi, Zeming Li, Jian Sun, and Jiaya Jia. Unifying voxel-based representation with transformer for 3d object detection. *arXiv preprint arXiv:2206.00630*, 2022. 6

[52] Dihe Huang, Ying Chen, Yikang Ding, Jinli Liao, Jianlin Liu, Kai Wu, Qiang Nie, Yong Liu, and Chengjie Wang. Rethinking dimensionality reduction in grid-based 3d object detection. *arXiv preprint arXiv:2209.09464*, 2022. 6

[53] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017. 5

[54] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34:24261–24272, 2021. 5

[55] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 6

[56] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 6

[57] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 6

[58] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. Torchsparse: Efficient point cloud inference engine. *Proceedings of Machine Learning and Systems*, 4:302–315, 2022. 6

[59] Spconv Contributors. Spconv: Spatially sparse convolution library. https://github.com/traveller59/spconv, 2022. 6

[60] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. Rangenet++: Fast and accurate lidar semantic segmentation. In *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 4213–4220. IEEE, 2019. 7

[61] Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. Squeeze-segv3: Spatially-adaptive convolution for efficient point-cloud segmentation. In *European Conference on Computer Vision*, pages 1–19. Springer, 2020. 7

[62] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. Salsanext: Fast, uncertainty-aware semantic segmentation of lidar point clouds. In *International Symposium on Visual Computing*, pages 207–222. Springer, 2020. 7

[63] Ran Cheng, Ryan Razani, Ehsan Taghavi, Enxu Li, and Bingbing Liu. 2-s3net: Attentive feature fusion with adaptive feature selection for sparse semantic segmentation network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12547–12556, 2021. 7

[64] Maosheng Ye, Shuangjie Xu, Tongyi Cao, and Qifeng Chen. Drinet: A dual-representation iterative learning network for point cloud segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7447–7456, 2021. 7

[65] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019. 6, 7

[66] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012. 7

[67] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4413–4421, 2018. 7