Not All Neighbors Matter: Point Distribution-Aware Pruning for 3D Point Cloud

Yejin Lee[1], Donghyun Lee[1], JungUk Hong[1], Jae W. Lee[1], Hongil Yoon[2]

[1]Seoul National University
[2]Google
yejinlee@snu.ac.kr, eudh1206@snu.ac.kr, junguk16@snu.ac.kr, jaewlee@snu.ac.kr, hongilyoon@google.com

Abstract

Applying deep neural networks to 3D point cloud processing has demonstrated a rapid pace of advancement in those domains where 3D geometry information can greatly boost task performance, such as AR/VR, robotics, and autonomous driving. However, as the size of both the neural network model and 3D point cloud continues to scale, reducing the entailed computation and memory access overhead is a primary challenge to meet strict latency and energy constraints of practical applications. This paper proposes a new weight pruning technique for 3D point cloud based on spatial point distribution. We identify that particular groups of neighborhood voxels in 3D point cloud contribute more frequently to actual output features than others. Based on this observation, we propose to selectively prune less contributing groups of neighborhood voxels first to reduce the computation overhead while minimizing the impact on model accuracy. We apply our proposal to three representative sparse 3D convolution libraries. Our proposal reduces the inference latency by 1.60× on average and energy consumption by 1.74× on NVIDIA GV100 GPU with no loss in accuracy metric.

1 Introduction

With the rapid advancements in capabilities of both computing and sensor (i.e., LiDARs, RGB-D cameras), 3D point cloud has become a popular representation of scenes and objects in domains such as AR/VR, robotics, and autonomous driving, where 3D geometry information can greatly boost the task performance. There is extensive research that applies deep neural networks to 3D point cloud to efficiently enable new machine learning-based use cases such as 3D segmentation, object detection, classification, reconstruction, registration, completion, pose estimation, etc.

To effectively extract useful information out of a 3D point cloud obtained from sensors, it is often voxelized (Xu et al., 2021b; Song 2016; Liang et al., 2019) to impose regularity on irregular 3D point cloud by partitioning the 3D space into structured grids of voxels. Unlike dense convolution, there are many nearby empty voxels in the 3D space and they do not actually contribute to generating output features. Carrying out the computation on them greatly impacts performance and energy consumption. With this sparse nature of 3D point cloud, convolution operations for 3D point cloud are known as *sparse 3D convolution*. There are several representative sparse 3D convolution libraries such as Spconv (Yan et al., 2018), MinkowskiEngine (Choy et al., 2019) and TorchSparse (Tang et al., 2022), to efficiently skip unnecessary computations on the empty voxels.

However, along with the rapid increase in the size of 3D point cloud neural network models and datasets, applications have demanded more computing power despite software optimization. Reducing such computational overhead is a primary challenge given that many practical use cases of 3D point cloud require strict latency and energy constraints.

To address this challenge, we propose a new *spatial point distribution-aware weight pruning technique* for 3D point cloud. Our empirical analysis finds that particular groups of neighborhood voxels in 3D point cloud contribute more frequently to actual output features than others. Our proposal leverages this observation to selectively prune neighborhood voxels by excluding groups of less contributing neighborhood voxels for each layer. With this approach, we can significantly reduce the computation and memory access cost required for sparse 3D convolution operations. We apply the proposed technique to the three representative sparse 3D convolution libraries. The results show a significant reduction in FLOPs and model parameter size. This translates to a considerable reduction in both end-to-end inference latency and energy consumption for various tasks.

Our key contributions are summarized as follows:

- We are the first to make an empirical observation that neighborhood voxels can be clustered into a small number of groups, where the voxels in a group have a similar probability of having a point. This non-uniform, tiered distribution of non-empty neighborhood voxels reflects the spatial point distribution of the real-world dataset.
- Leveraging this observation, we propose a novel spatial point distribution-aware pruning technique. Since not all layers in a model have the same sensitivity to pruning, we introduce a new metric called pruning-friendliness (PF) to guide layer-wise non-uniform settings of the degree of pruning. We also provide a search strategy to greatly reduce the search space of this pruning parameter setting.
- We apply our proposal to three libraries: Spconv, MinkowskiEngine and TorchSparse. Our evaluation on NVIDIA GV100 shows significant improvements in both

the overall latency by 1.60× and the energy efficiency by 1.74× on average with no accuracy loss.

2 Related Work

2.1 3D Point Cloud Neural Networks

Point-based Approach PointNet (Qi et al., 2016) first proposes applying deep neural networks to raw points in a 3D point cloud without voxelization. PointNet++ (Qi et al., 2017) adds hierarchy and grouping features to PointNet. Derivative works have been proposed as well (Qi et al., 2019; Ran et al., 2022; Zhang et al., 2020; Qian et al., 2022; Xiang et al., 2021; Wijaya et al., 2022). Point-based approaches find a set of input points to compute the feature vector for an output point with the k-nearest neighbors algorithm or ball query method and aggregate features of input points using MLP or max pooling or other operations. Because of lack of the voxelized representation support, the approaches show more irregular memory access patterns and are not a target of our work.

Voxel-based Approach This approach first converts 3D point cloud into a voxelized representation and operates on voxels rather than raw points. Each voxel either contains a single point that represents the voxel or is empty. Early voxel-based approaches process voxels with conventional CNNs (i.e., dense 3D convolution), demanding a large amount of computation and memory usage. However, the use of sparse 3D convolution (Graham et al., 2018) makes the approach more lightweight, hence harnessing their potential. Recently, voxel-based models have achieved state-of-the-art performance in various tasks to make sparse 3D convolution the key primitive in 3D point cloud processing. Subsequently, a number of voxel-based neural network models have been proposed (Nekrasov et al., 2021; Han et al., 2020; Hu et al., 2021a,b; Choy et al., 2019; Robert et al., 2022; Xu et al., 2021a; Zheng et al., 2021; Cheng et al., 2021; Rukhovich et al., 2022, 2021) to advance the state-of-the-art for those tasks.

2.2 Sparse 3D Convolution Libraries

Since sparse 3D convolution shows different properties from dense 3D convolution, standard machine learning frameworks, e.g., PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2015), do not support this operation. There are three representative libraries to optimize the operation. MinkowskiEngine (Choy et al., 2019) and TorchSparse (Tang et al., 2022) perform the sparse 3D convolution by repeating gather-matmul-scatter operations for nearby voxels. Spconv (Yan et al., 2018) modifies implicit GEMM (Chetlur et al., 2014) to make it suitable for the sparse 3D convolution. We evaluate our proposal on top of the three libraries in Section 5.

2.3 Sparse 3D Convolution Optimization

Cylinder3D (Zhu et al., 2020) is a model for a 3D semantic segmentation task, especially for extremely sparse outdoor 3D point cloud. In addition to its main contribution, this work proposes an optimization technique that could accelerate sparse 3D convolution by replacing existing residual block with Asymmetrical Residual Block (ARB), which reduces computation while maintaining the same receptive field. The original residual block consists of two 3×3×3 sized sparse 3D convolutions, but ARB consists of two $3 \times 1 \times 3$ and two $1 \times 3 \times 3$ resulting in total four sparse 3D convolution layers. This optimization does not reflect any dataset distribution unlike our proposal. We compare ARB with our proposal in Section 5.

There is a work that proposes a 4D convolution neural network for 3D-video perception, which takes both space (3D) and time (1D) into consideration (Choy et al., 2019). However, the computation and memory cost increase exponentially in the 4D space due to the curse of dimensionality. Thus, along with the 4D neural network, they propose a hybrid kernel that applies traditional kernel shape along spatial dimensions (3D) for accurate geometry information and a cross-shaped kernel along the temporal dimension (1D). On the other hand, our proposal focuses on how to effectively prune kernels in the spatial dimensions. Since our proposal is orthogonal to the hybrid kernels, we can expect a synergy when deployed together.

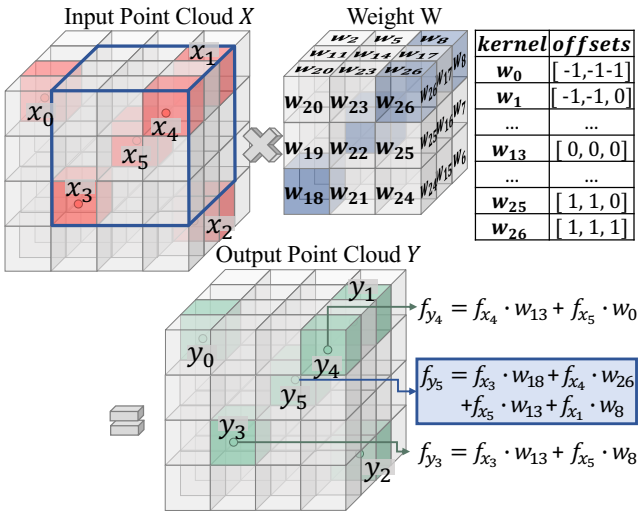


Figure 1: Sparse 3D Convolution Operation.

3 Background: Sparse 3D Convolution

Sparse 3D Convolution (Graham et al., 2018) is an operation that takes an input point cloud X as an input to compute output point cloud Y. When the input channel count of the convolution layer is IC, an input point $x_i \in X$ has a feature vector $f_{x_i} \in R^{IC}$ and a coordinate $c_{x_i} \in R^D$ in D-dimensional vector space. Similarly, when the output channel count of the convolution layer is OC, an output point $y_j \in Y$ has a feature vector $f_{y_j} \in R^{OC}$ and a coordinate $c_{y_j} \in R^D$. Given a kernel size K, this operation has weight $W \in R^{K^D \times IC \times OC}$. Weight W consists of K^D matrices where each matrix's size is $IC \times OC$. Here, each weight matrix (or kernel) w_k gets a unique *offsets*$[k]$

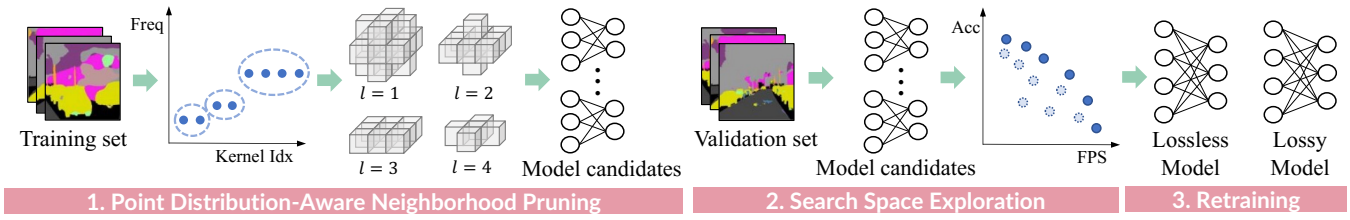


Figure 2: Overall Flow of Our Proposal.

in $\{\frac{-(K-1)}{2}, ..., \frac{(K-1)}{2}\}^D$ as shown in the bottom left table of Figure 1.[1] Finally, this operation also takes parameter s which denotes the stride length. Formally, the operation is described as follows.

$$f_{y_j} = \sum_{x_i \in X} \sum_{k=0..K^D-1} f_{x_i}\, w_k\, p_j[k], \text{ where } y_j \in Y$$
$$p_j[k] = \begin{cases} 1 & \text{if } c_{x_i} = s \cdot c_{y_j} + \textit{offsets}[k] \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Figure 1 exemplifies a sparse 3D convolution operation for the output point y_5 when $K = 3$, $s = 1$ and $D = 3$. Since the number of input and output points remains the same when stride $s = 1$, both input and output point cloud X and Y have six points (colored with orange/mint). Weight W consists of $K^D = 3^3$ kernels and kernels are indexed from 0 to $K^D - 1$. To compute the feature vector for output point j, the set of input points (i.e., *neighbors*) in the receptive field (i.e., *neighborhood*) is first identified by iterating over neighborhood voxels and checking if $p_j[k] = 1$. $p_5[18] = 1$ since $c_{x_3} = c_{y_5} + \textit{offsets}[18]$. Thus, the input point x_3 is added to a neighbor set. Likewise, input point x_4, x_5 and x_1 are added to the neighbor set. Then, feature vectors in the neighbor set $f_{x_3}, f_{x_4}, f_{x_5}, f_{x_1}$ are multiplied with the corresponding kernel matrices $w_{18}, w_{26}, w_{13}, w_8$ and summed up to compute the output feature vector f_{y_5} for output point 5.

In summary, the key idea of sparse 3D convolution is to find *neighbors* in *neighborhood* voxels of size K^D and utilize neighbors' feature vectors to compute the feature vector of the output point at the center of the neighborhood. There could be up to K^D neighbors; however, it is a very rare case in 3D point cloud due to its sparse nature.

4 Spatial Point Distribution-Aware Neighborhood Pruning

We propose a technique that effectively prunes neighborhood voxels based on the metric that captures spatial point distribution of a given dataset. The design objective is to accelerate inference latency with minimal impact on the model accuracy metric. Figure 2 shows the overall flow of our proposal. ❶ We first analyze neighborhood voxels in 3D space for each layer based on train sets and cluster them based on the analysis. Here, pruning is performed at the granularity of clusters, and each layer can take a different degree of pruning. ❷ Then, we explore the search space across layers to identify an optimal configuration that maximizes the speedup with minimal effect on the accuracy metric. Since the search space grows exponentially to the number of layers, we introduce a new search strategy to reduce the search space. ❸ Finally, with a selected configuration, we perform retraining to recover losses from the neighborhood pruning.

4.1 Analysis of Neighborhood Voxels

The rationale behind our approach is that the chances of having a neighbor (i.e., a point in the neighborhood) differ for each neighborhood voxel in the 3D space. The chances are dependent on spatial point distribution in neighborhood voxels for a given 3D point cloud. They are also a measure of how strongly the corresponding voxels contribute to the value of the output feature vector. Thus, understanding the chances by considering the spatial point distribution is critical for our proposal.

We define the probability of having a neighbor in a given neighborhood voxel k for the training set T as follows, where $center$ denotes the neighborhood voxel (kernel) index where output point y_j resides.

$$Prob(p_j[k] = 1) = \frac{\sum_{Y \in T} \sum_{y_j \in Y} p_j[k]}{\sum_{Y \in T} \sum_{y_j \in Y} p_j[center]} \quad (2)$$

Figure 3 shows the probability of having a neighbor for each neighborhood voxel for the $7th$ sparse 3D convolution layer of MinkUNet model on the SemanticKITTI training set. We observe that 1) certain neighborhood voxels have higher chances of having neighbors and 2) there are more popular groups of neighborhood voxels in which neighbors appear more frequently. For example, neighborhood voxels such as 24, 2, 0, 26, 6, 20, 18 and 8 are relatively farther from the center voxel where the output point resides and have substantially less chance of having neighbors (e.g., $<$19%). On the other hand, neighborhood voxels 4, 22, 16 and 10 are nearby sharing the same X, Y axis coordinate with the center voxel and have a higher chance of having neighbors (e.g., $>$45%). We observe that the probability of having neighbors in a neighborhood voxel differs not only across the layers in the model but also across datasets.

Neighbor voxels with high probability of having neighbors imply that points representing object or scene are often placed along the direction of corresponding voxels. Conceptually, we compress spatial point distribution information

[1] The one in the text is based on an odd number K. With an even number K, three libraries, i.e., MinkowskiEngine, TorchSparse and Spconv, have different definitions for the *offsets*. Either *offsets* $= \{\frac{-K}{2}, ..., \frac{(K-2)}{2}\}^D$ or *offsets* $= \{\frac{-(K-2)}{2}, ..., \frac{K}{2}\}^D$ is selected.

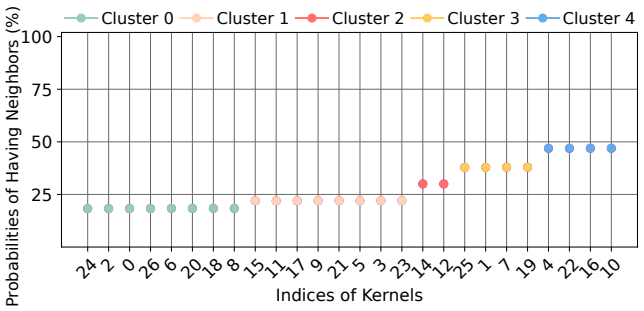


Figure 3: Probability of Having a Neighbor. X-axis indicates kernel indices of voxels shown in Figure 1, and the data is sorted in ascending orders in terms of the probability. We used randomly sampled 3826 frames from train dataset.

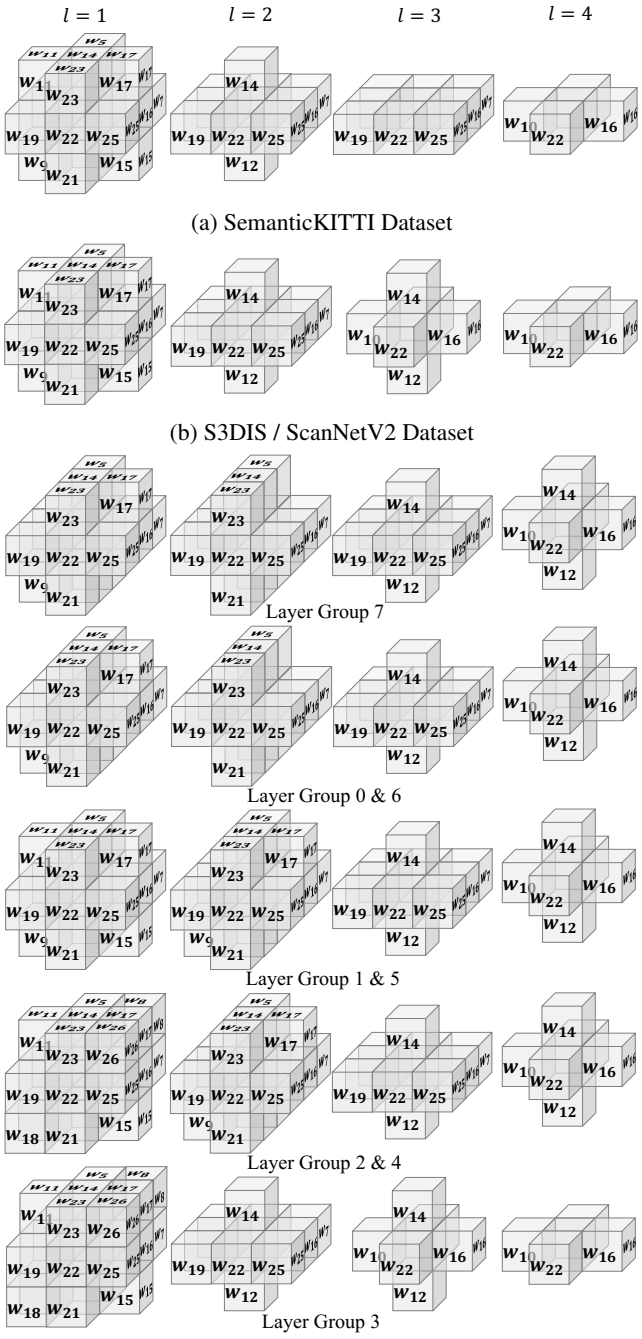
into K^D probabilities of the neighborhood voxels. *The empirical analysis implies that the probability of having neighbors is an effective metric for reflecting dataset property and determining which neighborhood voxels to prune.*

Based on this observation, we propose to prune neighborhood voxels (kernels) for sparse 3D convolution by exploiting the spatial point distribution information. As discussed before, there are several neighborhood voxels having a similar chance of containing neighbors, we first cluster them. Then, starting from the cluster of voxels having the lowest probability of having neighbors, we incrementally exclude the next cluster of voxels as the degree of pruning goes up.

Clustering Neighborhood Voxels The primary goal of clustering is to group neighborhood voxels whose chances of having neighbors are similar. By pruning a cluster of neighborhood voxels together, we can greatly reduce the search space for layer-wise pruning. To cluster all neighborhood voxels into M clusters, the algorithm first sorts the neighborhood voxels according to their chance of containing neighbors (Refer to Figure 3). Then, delta values between the current neighborhood voxel and the next neighborhood voxel are computed for all sorted voxels. Then, the algorithm identifies the top $M-1$ delta values to set the boundaries between adjacent clusters. Figure 3 visualizes the clustering result, the five clusters are shown in distinct colors. The top 4 neighborhood voxels with the highest probability (i.e., Cluster 10,16,22 and 4) are guaranteed not to be pruned to ensure that the neighborhood includes a minimum number of voxels that most frequently contribute to actual output features.

Figure 4 illustrates four types of the neighborhood shape based on the five clusters by incrementally excluding clusters with the least probability of having neighbors for SemanticKITTI, S3DIS, ScanNetV2 and Sun RGB-D datasets for sparse 3D convolution layer groups of MinkUNet model used in the evaluation. Note that Figure 4(a) visualizes the neighborhood shape based on the information in Figure 3.

The shape of pruned neighborhood reflects the spatial point distribution of the dataset. SemanticKITTI is a dataset whose data are collected from LiDAR sensors of self-driving cars. Its spatial range in a scene is much larger and it is also more sparse than other 3D indoor point cloud datasets.



(a) SemanticKITTI Dataset

(b) S3DIS / ScanNetV2 Dataset

(c) Sun RGB-D Dataset

Figure 4: Illustration of Pruned Neighborhood for Sparse 3D Convolution Layer Groups of MinkUNet Model. SemanticKITTI, S3DIS and ScanNetV2 show the same clustering result across all sparse 3D convolution layer groups, while Sun RGB-D shows different set of clustering result.

It is known that surface information is important for self-driving cars (Behley et al., 2019); thus, SemanticKITTI contains lots of points on the surface. For this reason, points in the sparse region of the scene often do not have points in above and below the horizontal neighborhood voxels. Thus,

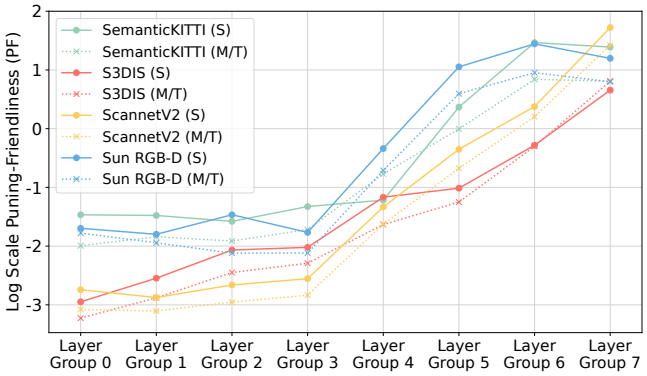


Figure 5: Log-scale PF for Each Layer Group in the Models of Four Datasets. S, M, T stand for Spconv, MinkowskiEngine, and TorchSparse, respectively

the shape at $l = 3$ is a horizontal plane reflecting such characteristics in SemanticKITTI. On the other hand, Sun RGB-D is an indoor dataset for detection models, which is relatively denser and its voxel size (2.5cm) is smaller than that of SemanticKITTI (5cm). Points in this dataset often represent objects. The difference resulting from spatial point distribution of datasets are reflected in the clustering results and possible pruned neighborhood shapes. Figure 4c shows more dense pruned neighborhood shapes at a given degree of pruning (l=2-4).

4.2 Search Space Exploration and Retraining

It is well known that different layers in neural network models have different levels of tolerance to errors. To fully exploit such characteristics, we choose to include different numbers of clusters in the neighborhood across different layers. The objective here is to find the optimal value of l_i for each layer i in the neural network model, where l_i represents the number of clusters excluded from the neighborhood whose chance of having neighbors is the lowest. Note that l_i ranges from 0 to $M-1$ and higher l_i means pruning more neighborhood voxels.

However, assuming there are N sparse 3D convolution layers in the model, a brute force search will require exploring M^N configurations which are infeasible. For instance, assuming $M = 5$ and the number of layers in the model as $N = 40$, the number of available combinations is $l^N = 5^{40} \approx 9.094947e + 27$. Thus, we propose a search strategy to reduce the search space, which works as follows.

First, we reduce the search space by grouping consecutive layers whose neighbor distributions are identical and applying the same degree of neighborhood pruning for the layers. From now on, we redefine N as the number of layer groups and define $L = [l_0, ..., l_{N-1}]$ where l_i refers to a degree of neighborhood pruning for the i^{th} layer group.

Then, we evaluate each layer group's sensitivity to neighborhood pruning. Specifically, we compute each layer group's Pruning-Friendliness (PF) which is defined as $flops\_reduction \div accuracy\_loss$ when $l_i = M-1$ neighborhood shape is applied to only the given layer group. Note that $accuracy\_loss$ is the absolute value of accuracy loss from the model's baseline performance without retraining. A higher value indicates the layer group has a relatively smaller accuracy loss from neighborhood pruning while reducing a large amount of computation, making it a good target for neighborhood pruning.

Figure 5 reports PF values for all layer groups in the models of four datasets used in our evaluation. X-axis represents layer groups and Y-axis represents log scale PF values, i.e., $log(PF)$, to clearly show the difference among layer groups. The target model is called MinkUNet and it consists of eight layer groups and the solid line represents Spconv and the dotted line represents MinkowskiEngine and TorchSparse. How Spconv performs sparse 3D convolution layers is slightly different from how MinkowskiEngine and TorchSparse perform them (Refer to Section 5.2 and 5.3). This leads to differences in FLOPs requirement for processing an identical sparse 3D convolution layer, and thus corresponding PF values are also different.

Our search strategy allows us to explore only configurations where l_i of a more pruning-friendly layer group i is always greater than or equal to l_j of a less pruning-friendly layer group j. For example, when we consider layer groups of Sun RGB-D dataset with Spconv library, our search strategy only explores configurations where $l_6 >= l_7 >= l_5 >= l_4 >= l_2 >= l_0 >= l_3 >= l_1$, following the descending order of corresponding PF values. Refer to Table 1 for the selected configurations. This effectively reduces the search space from M^N to ${}_MH_N = {}_{M+N-1}C_N$. When $N = 8$ and $M = 5$, such a strategy reduces the search space from 390,625 (5^8) to 495 (${}_{5+8-1}C_8$).

To further reduce the search space, our search strategy also evaluates a configuration's accuracy on the validation set. Specifically, when a configuration's accuracy is below the threshold t, we simply skip evaluating more aggressive configurations than the current configuration. Here, a configuration a is more aggressive than a configuration b if $L_a[i] >= L_b[i],\ \forall i \in \{0, ..., N-1\}$.

Algorithm 1 elaborates our search strategy when $N = 4$ and $M = 4$. First, it starts from the least aggressive configuration $L = [0, 0, 0, 0]$ (Line 1). Note that elements in L are in order of layer groups sorted by descending PF values so that layer groups with higher PF values are placed before layer groups with lower PF values. Thus, the nth number in L represents the corresponding l for nth most pruning-friendly layer group. After evaluating the current configuration on validation set (Line 6), we move on to the next configuration (Line 10-16). Assuming that the accuracy for the current configuration is above t, configurations $[1, 0, 0, 0]$ followed by $[1, 1, 0, 0]$ are evaluated if both configurations' accuracy is still above the t. This process is continued till it reaches $[1, 1, 1, 1]$. If this configuration's accuracy is still above t, we continue with configuration $[2, 0, 0, 0]$ followed by $[2, 1, 0, 0]$. Assume that our search strategy finds that the accuracy of the configuration $[2, 1, 1, 0]$ is below t (Line 7). This indicates that the configuration $[2, 1, 1, 0]$ is already too aggressive, and thus we skip exploring configurations that are strictly more aggressive than this configuration (Line 3-4) (e.g., $[2, 1, 1, 1]$, $[2, 2, 1, 0]$, etc.). We resume searching

Algorithm 1: Configuration Search Strategy Algorithm

Parameter: N: number of layers, t: accuracy lower bound, M: number of clusters

1: $L = [0] * N$, $L' = L$, skip = False
2: **while** all($l < M$ for l in L) **do**
3: **if** skip and any($l' > l$ for l',l in zip(L',L)) **then**
4: skip = False
5: **if** skip is False **then**
6: val_acc = Validation with L
7: **if** val_acc $< t$ **then**
8: skip = True, $L' = L$
9: // Move on to the next configuration
10: idx = 0
11: **for** $i \leftarrow |L| - 1$ to 0 **do**
12: **if** $L[i-1] > L[i]$ **then**
13: idx = i
14: **break**
15: L[idx]++
16: L[idx+1:] = 0

from $[3, 0, 0, 0]$. This process is repeated until we reach the most aggressive configuration which is $[3, 3, 3, 3]$ (Line 2).

Once the exploration finishes, we identify pareto frontier configurations in terms of accuracy loss versus speedup on the target device. Then it performs retraining with neighborhood pruning configuration that achieves the most speedup among them. If this configuration fails to reach the original accuracy, we check a less aggressive configuration with the next largest speedup. If a configuration achieving the original accuracy is found, the selected configuration is used.

5 Evaluation

5.1 Methodology

Dataset We test four datasets to show the effectiveness of our proposal for various 3D point cloud usecases. For 3D semantic segmentation tasks, we use an outdoor dataset SemanticKITTI (Behley et al., 2019) as well as two indoor datasets S3DIS (Armeni et al., 2016) and ScanNetV2 (Dai et al., 2017). We use an indoor dataset Sun RGB-D (Song et al., 2015) for 3D object detection tasks. The average number of input points per frame for the validation set are 87733, 50874, 109603, and 9800, respectively. We utilize widely used voxel size, 5cm for SemanticKITTI and Stanford, 2cm for ScanNetV2 and 2.5cm for Sun RGB-D.

Models and Metrics For 3D semantic segmentation task, we use MinkUNet (Choy et al., 2019) which is one of the most representative model using sparse 3D convolution. For 3D object detection task, we use a combined model (Hou et al., 2021; Xie et al., 2020) that has MinkUNet as a backbone module and integrates modified VoteNet (Qi et al., 2019) to output bounding box coordinates. Also, we use mIOU (mean Intersection Over Union) as an accuracy metric for 3D semantic segmentation and mAP@0.25 (mean Average Precision) as an accuracy metric for 3D object detection. Since test set labels are not provided, we report the final accuracy on validation set.

Dataset	Library	Lossless	Lossy
Semantic KITTI	S	[0,0,0,0,0,2,4,4]	[0,0,0,0,0,4,4,4]
	M & T	[0,0,0,0,1,1,4,4]	[0,0,0,0,0,4,4,4]
S3DIS	S	[0,0,0,0,0,0,0,4]	[0,0,1,1,1,1,3,4]
	M & T	[0,0,0,0,0,0,0,4]	[0,0,1,1,1,1,3,4]
ScanNetV2	S	[0,0,0,0,0,0,1,3]	[0,0,0,0,0,3,4,4]
	M & T	[0,0,0,0,0,0,1,3]	[0,0,0,0,0,3,4,4]
Sun RGB-D	S	[0,0,1,0,1,4,4,4]	[2,0,2,1,2,4,4,4]
	M & T	[1,0,0,0,1,1,4,3]	[1,1,0,0,2,3,4,4]

Table 1: Selected Configurations for Each Library. S, M, T stands for Spconv, MinkowskiEngine, and TorchSparse, respectively. A configuration consists of values for eight corresponding layer groups, and each value indicates one of $l = 0, 1, 2, 3, 4$ from Figure 4 for a layer group.

Implementation Details and Environments We compute $Prob(p_j[k] = 1)$ of Equation 2 for individual layers of a given model based on its train set frames. Since SemanticKITTI has much more train set frames (19130) than others, 20% of the train set frames (3826) are randomly sampled and used. We evaluate both lossless and lossy configurations. For the lossless configuration, we choose a configuration that achieves the maximum speedup and also reaches or exceeds the baseline accuracy metric after retraining. For the lossy configuration, we choose a configuration that achieves the maximum speedup and shows 1% lower accuracy metric at maximum after retraining.

Our target models have eight sparse 3D convolution layers for upsampling/downsampling, and each layer is followed by two residual blocks (each residual block has two sparse 3d convolution layers), making it total sixteen residual blocks. We group consecutive layers into eight layer groups whose neighbor distributions are identical. As a result, each layer group has the same clustering result; thus, we applied different l for each layer group. Additionally, they have one more spare 3D convolution layer at the beginning of the model, but we exclude it from the pruning. Our search strategy explores possible configurations $L = [l_0, ..., l_7]$ that follows our constraints on PF and threshold t. The accuracy threshold t is set to -20% below the original validation accuracy. For Sun RGB-D dataset on Spconv library, t is set to -30% below the original validation accuracy. Table 1 summarizes selected configurations. The selected configuration for the same dataset may differ across the libraries since the order of each layer group's PF differs.

We apply our proposal to Spconv v.2.1.21, MinkowskiEngine v0.5.4, and TorchSparse v.1.4.0 in CUDA. We use PyTorch 1.8.1, CUDA 11.2 for the setup. We used NVIDIA GPU GV100 with 32GB memory capacity and FP32 datatype for all results in this section.

5.2 Pruning Impacts

FLOPs Table 2a presents the FLOPs reduction with the neighborhood pruning for lossless and lossy configurations. We consider MAC operation as a single floating point operation and exclude FLOPs required for layers other than sparse 3D convolution. Our proposal significantly reduces the total

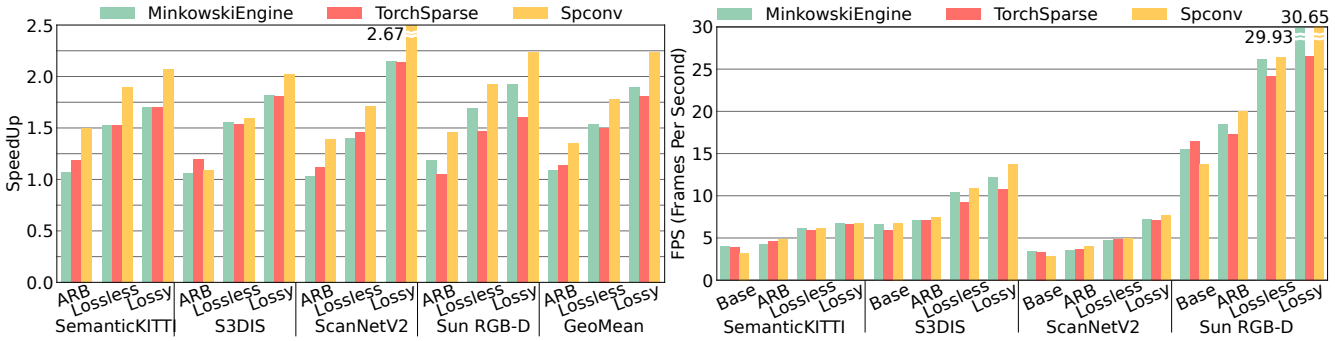
Dataset	Lib-rary	Lossless			Lossy		
		FLOPs ↓ (%)	Neighbor ↓ (%)	Model param↓ (%)	FLOPs ↓ (%)	Neighbor ↓ (%)	Model param↓ (%)
Semantic KITTI	S	54.77	25.08	45.48	59.60	27.96	50.10
	M&T	40.69	23.53	45.73	48.39	27.96	50.10
S3DIS	S	46.61	31.01	16.59	61.29	39.15	47.09
	M&T	45.47	31.28	16.59	56.91	39.30	47.09
Scannet V2	S	48.21	25.34	21.11	71.67	41.55	48.56
	M&T	39.21	25.33	21.11	64.42	41.55	48.56
Sun RGB-D	S	81.04	45.28	58.23	82.13	54.06	64.66
	M&T	55.98	39.45	44.28	70.52	50.23	59.89

(a) FLOPs, Number of Neighbors and Model Parameter Reduction.

Base-line	ARB	Lossless	Lossy
63.13	57.50	63.34	62.41
62.11		62.22	61.12
63.73	62.27	64.11	62.73
64.06		64.07	63.29
72.35	70.00	72.41	71.46
72.34		72.38	71.34
57.06	52.60	57.21	56.34
56.23		56.52	55.69

(b) Model Performance.

Table 2: Impact of Our Proposal. S, M, T stand for Spconv, MinkowskiEngine, and TorchSparse, respectively.



(a) Speedup Normalized to Each Library's Baseline

(b) FPS (Frames Per Second)

Figure 6: End-to-end Result for FP32 on GV100.

FLOPs of a target model. We see geomean 49.45% reduction for lossless configuration and 62.93% reduction for lossy configuration. The FLOPs reduction for Spconv is greater than that of MinkowskiEngine or TorchSparse. Unlike these two libraries, Spconv performs sparse 3D convolution utilizing a modified version of implicit GEMM (Chetlur et al., 2014). This well-known approach highly pipelines computation and memory operations. However, there exists unavoidable zero computations causing more FLOPs than other libraries. For this reason, Spconv could benefit more from our proposal because there are high chance of removing zero computation together by neighborhood pruning.

Number of Neighbors and Model Parameters Our proposal significantly reduces the total number of neighbors participating in sparse 3D convolution. The neighbor reduction in Spconv is slightly different from others for the same configuration as the methodology of generating output point cloud coordinates when $s > 1$ is different. Our proposal reduces the number of model parameters by 33.64%, 52.01% for lossless and lossy configuration, respectively. The model parameter reduction does not align with FLOPs reduction because the number of points is also a contributing factor to the FLOPs reduction. The amounts of reduction in FLOPs, neighors and model parameters differ across datasets since each dataset has different clustering results, and the search space exploration also yields different results in terms of the number of pruned neighborhood voxels across layers.

Model Performance Table 2b shows the model performance for baseline, lossless, and lossy configurations. For the lossless configurations, our proposal is able to fully recover the performance in terms of the accuracy metric loss. We observe that some cases even achieve above the baseline performance via retraining. For the lossy configurations, we also observe that the proposed technique shows less than 1% of the accuracy metric loss relative to the baseline performance by retraining.

5.3 Performance Improvement

Inference Latency Improvement Figure 6a shows end-to-end speedup for both lossless and lossy configurations. The proposed technique achieves geomean speedup 1.60× across the three libraries for the lossless configuration and 1.97× for the lossy configuration. The speedup translates to the improvement in FPS (Refer to Figure 6b).

The results substantiate that Spconv benefits the most from our proposal. Its significant FLOPs reduction translates to the most speedup among three libraries. Furthermore, we observe cases where Spconv outperforms the other two libraries with our pruning technique. For example, Spconv's baseline FPS is lower than others for ScanNetV2. However, augmented with our proposal, Spconv achieves higher FPS than MinkowskiEngine for lossless and lossy configurations.

Inference Latency Breakdown We measure the end-to-end latency breakdown of three libraries in Figure 7. Each

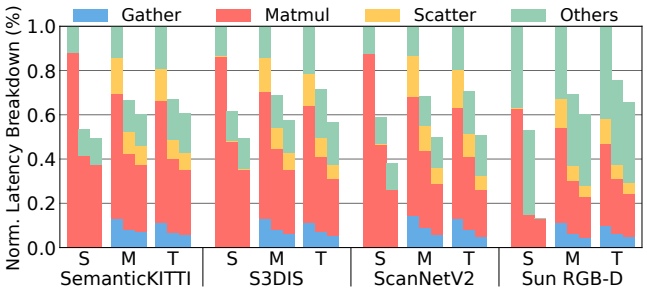


Figure 7: Normalized Latency Breakdown for Each Library.

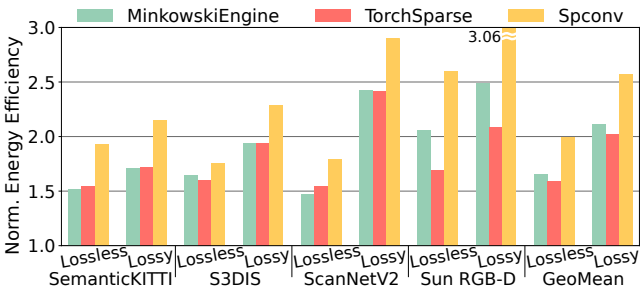


Figure 8: Normalized GPU Energy Efficiency.

library has three bars: baseline (without neighborhood pruning), lossless, and lossy configurations. Sub-components of each library are slightly different because of the differences in how to perform the sparse 3D convolution.

MinkowskiEngine and TorchSparse conduct three subsequent operations: gather, matmul, and scatter. First, the gather phase requires memory operations that collect input feature vectors to make sure computation only happens to non-empty neighborhood voxels. Second, the matmul phase performs matrix multiplications between gathered feature vectors and corresponding kernels. Lastly, the scatter phase requires memory operations to scatter outcomes to corresponding locations. Before pruning neighborhood, memory operations (i.e., gather and scatter) take an average of 27% and the matmul operations take 51%. *Others* portion includes latency required for layers other than sparse 3D convolution layers. This portion also includes preparation time in sparse 3D convolution layers such as metadata generation time and GPU kernel launch time for gather, matmul and scatter. Applying our proposal reduces the amount of gathered feature vectors, which in turn reduces the amount of downstream matmul computations and memory accesses for scatter operations. Thus, our proposal covers 78% of the total latency for optimization. The latency reduction in all three phases covered contributes to the considerable overall latency reduction for both lossless and lossy configurations.

Unlike above two libraries, Spconv performs memory and computation operations in a highly pipelined manner. It does not require explicit gather and scatter phases. Thus, it consists of only two subbars: matmul and others. The matmul operations account for 81% of the baseline end-to-end inference latency. Our proposal optimizes only the matmul part in Spconv. The results show significant reduction in the overall matmul latency across datasets.

Comparison with Asymmetric Residual Block Figure 6a also presents the speedup for models with Asymmetric Residual Blocks (ARB) (Zhu et al., 2020). The speedup benefit is relatively marginal relative to our approach. As discussed in Section 2, ARB doubles the number of sparse 3D convolution layers. Although the replaced layers require less computation relative to original residual blocks, the increase in the number of layers offsets the speedups from the replaced layers. This is particularly a severe problem on datasets with the small number of points (i.e., Sun RGB-D) since the preparation time overhead (*Others* portion in Figure 7) of sparse 3D convolution is relatively large and its time increases proportional to the number of layers. Furthermore, as shown in Table 2b, the use of ARB fails to achieve the baseline accuracy metrics for four datasets while our proposal achieves significantly higher speedup with minimal loss in the accuracy metric.

5.4 Energy Efficiency Improvement

For energy consumption analysis, we first measure GPU power consumption via `nvidia-smi` at 1ms intervals and calculate the average power consumption. We exclude the idle period before executing an inference. Then, we multiply the average GPU power with the total end-to-end inference time in seconds to compute the energy consumption. Figure 8 shows significant energy efficiency improvement for three libraries on lossless and lossy configurations. Our proposal improves energy consumption by 1.74×, 2.22× for lossless and lossy configurations, respectively.

5.5 Analysis of FP16 Datatype Models

Due to the high compute capability of FP16 tensor cores, the breakdown result shows slightly different behaviors. The portion of *Matmul* from Figure 7 decreases while the portion of *Others* increases compared to the FP32 results. As a result, the coverage of our proposal reduces. We achieve geomean end-to-end inference latency improvement by 1.39×, 1.61× and energy efficiency improvement by 1.30×, 1.47× for lossless and lossy configurations, respectively.

6 Conclusion

Sparse 3D convolution operation is one of the key operations in processing 3D point cloud. With the sparse nature of 3D point cloud, we observe that particular groups of neighborhood voxels contribute more frequently to actual output features than others. Our empirical analysis suggests that the proposed spatial point distribution-aware weight pruning technique is promising for 3D point cloud neural network models. Our proposal achieves substantial improvement in both real-time speedup and energy consumption by pruning groups of less contributing neighborhood voxels for each layer. We believe our work can enable 3D point cloud neural networks to be employed by a broader range of applications whose stringent latency and energy constraints would otherwise be difficult to satisfy. The source code is available at https://github.com/SNU-ARC/NotAllNeighborsMatter.git.

Acknowledgments

This work was supported by a research grant from Samsung Advanced Institute of Technology and Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (Developing Software Platform for Programming of PIM (2021-0-00853), Development of Model Compression Framework for Scalable On-device AI Computing on Edge Applications (2021-0-00105)). Jae W. Lee and Hongil Yoon are the corresponding authors.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Armeni, I.; Sener, O.; Zamir, A. R.; Jiang, H.; Brilakis, I.; Fischer, M.; and Savarese, S. 2016. 3D Semantic Parsing of Large-Scale Indoor Spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Behley, J.; Garbade, M.; Milioto, A.; Quenzel, J.; Behnke, S.; Stachniss, C.; and Gall, J. 2019. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Cheng, B.; Sheng, L.; Shi, S.; Yang, M.; and Xu, D. 2021. Back-tracing Representative Points for Voting-based 3D Object Detection in Point Clouds. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; and Shelhamer, E. 2014. cuDNN: Efficient Primitives for Deep Learning. *ArXiv*, abs/1410.0759.

Choy, C.; Gwak, J.; and Savarese, S. 2019. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dai, A.; Chang, A. X.; Savva, M.; Halber, M.; Funkhouser, T.; and Nießner, M. 2017. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Graham, B.; Engelcke, M.; and van der Maaten, L. 2018. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Han, L.; Zheng, T.; Xu, L.; and Fang, L. 2020. OccuSeg: Occupancy-Aware 3D Instance Segmentation. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hou, J.; Graham, B.; Nießner, M.; and Xie, S. 2021. Exploring data-efficient 3d scene understanding with contrastive scene contexts. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hu, W.; Zhao, H.; Jiang, L.; Jia, J.; and Wong, T.-T. 2021a. Bidirectional Projection Network for Cross Dimensional Scene Understanding. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hu, Z.; Bai, X.; Shang, J.; Zhang, R.; Dong, J.; Wang, X.; Sun, G.; Fu, H.; and Tai, C.-L. 2021b. VMNet: Voxel-Mesh Network for Geodesic-Aware 3D Semantic Segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Liang, M.; Yang, B.; Chen, Y.; and Hu, R. 2019. Multi-Task Multi-Sensor Fusion for 3D Object Detection. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Nekrasov, A.; Schult, J.; Litany, O.; Leibe, B.; and Engelmann, F. 2021. Mix3D: Out-of-Context Data Augmentation for 3D Scenes. In *Proceedings of the International Conference on 3D Vision (3DV)*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*.

Qi, C. R.; Litany, O.; He, K.; and Guibas, L. J. 2019. Deep Hough Voting for 3D Object Detection in Point Clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Qi, C. R.; Su, H.; Mo, K.; and Guibas, L. J. 2016. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *ArXiv*, abs/1612.00593.

Qi, C. R.; Yi, L.; Su, H.; and Guibas, L. J. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *ArXiv*, abs/1706.02413.

Qian, G.; Li, Y.; Peng, H.; Mai, J.; Hammoud, H. A. A. K.; Elhoseiny, M.; and Ghanem, B. 2022. PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*.

Ran, H.; Liu, J.; and Wang, C. 2022. Surface Representation for Point Clouds. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Robert, D.; Vallet, B.; and Landrieu, L. 2022. Learning Multi-View Aggregation In the Wild for Large-Scale 3D Se-

mantic Segmentation. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Rukhovich, D.; Vorontsova, A.; and Konushin, A. 2021. FCAF3D: Fully Convolutional Anchor-Free 3D Object Detection. *ArXiv*, abs/2112.00322.

Rukhovich, D.; Vorontsova, A.; and Konushin, A. 2022. Imvoxelnet: Image to voxels projection for monocular and multi-view general-purpose 3d object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*.

Song, S.; Lichtenberg, S. P.; and Xiao, J. 2015. SUN RGB-D: A RGB-D scene understanding benchmark suite. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Song, S.; and Xiao, J. 2016. Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tang, H.; Liu, Z.; Li, X.; Lin, Y.; and Han, S. 2022. TorchSparse: Efficient Point Cloud Inference Engine. In *Proceedings of Machine Learning and Systems (MLSys)*.

Wijaya, K. T.; Paek, D.-H.; and Kong, S.-H. 2022. Advanced Feature Learning on Point Clouds using Multi-resolution Features and Learnable Pooling. *ArXiv*, abs/2205.09962.

Xiang, T.; Zhang, C.; Song, Y.; Yu, J.; and Cai, W. 2021. Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Xie, S.; Gu, J.; Guo, D.; Qi, C.; Guibas, L. J.; and Litany, O. 2020. PointContrast: Unsupervised Pre-training for 3D Point Cloud Understanding. *ArXiv*, abs/2007.10985.

Xu, Q.; Zhou, Y.; Wang, W.; Qi, C. R.; and Anguelov, D. 2021a. Spg: Unsupervised domain adaptation for 3d object detection via semantic point generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Xu, Y.; Tong, X.; and Stilla, U. 2021b. Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry. *Automation in Construction*.

Yan, Y.; Mao, Y.; and Li, B. 2018. Second: Sparsely embedded convolutional detection. *Sensors*.

Zhang, Z.; Sun, B.; Yang, H.; and Huang, Q. 2020. H3dnet: 3d object detection using hybrid geometric primitives. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Zheng, W.; Tang, W.; Jiang, L.; and Fu, C.-W. 2021. SE-SSD: Self-Ensembling Single-Stage Object Detector From Point Cloud. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhu, X.; Zhou, H.; Wang, T.; Hong, F.; Ma, Y.; Li, W.; Li, H.; and Lin, D. 2020. Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation. *ArXiv*, abs/2011.10033.