PVNAS: 3D Neural Architecture Search with Point-Voxel Convolution

Zhijian Liu*, Haotian Tang*, Shengyu Zhao, Kevin Shao, and Song Han

Abstract—3D neural networks are widely used in real-world applications (*e.g.*, AR/VR headsets, self-driving cars). They are required to be fast and accurate; however, limited hardware resources on edge devices make these requirements rather challenging. Previous work processes 3D data using either voxel-based or point-based neural networks, but both types of 3D models are not hardware-efficient due to the large memory footprint and random memory access. In this paper, we study 3D deep learning from the efficiency perspective. We first systematically analyze the bottlenecks of previous 3D methods. We then combine the best from point-based and voxel-based models together and propose a novel hardware-efficient 3D primitive, *Point-Voxel Convolution (PVConv)*. We further enhance this primitive with the sparse convolution to make it more effective in processing large (outdoor) scenes. Based on our designed 3D primitive, we introduce *3D Neural Architecture Search (3D-NAS)* to explore the best 3D network architecture given a resource constraint. We evaluate our proposed method on six representative benchmark datasets, achieving state-of-the-art performance with **1.8-23.7**× measured speedup. Furthermore, our method has been deployed to the autonomous racing vehicle of MIT Driverless, achieving larger detection range, higher accuracy and lower latency.

Index Terms—3D Point Cloud, Neural Architecture Search, Efficient Deep Learning, Autonomous Driving.

✦

1 INTRODUCTION

3D deep learning has received increased attention thanks to its wide applications. It has been applied in AR/VR headsets to understand the layout of indoor scenes; it has also been used in the LiDAR perception that serves as the eyes of autonomous driving systems to understand the semantics of outdoor scenes to parse the drivable area (*e.g.*, roads, parking areas). These real-world applications require high accuracy and low latency at the same time: *i.e.*, AR/VR headsets aim to offer an instant and accurate response for better user experience, and self-driving cars are expected to drive safely even at a relatively high speed. However, the computational resources on these devices are tightly constrained by the form factor (since we do not want a full backpack of hardware or a whole trunk of workstations) and heat dissipation. Thus, it is crucial to design efficient and effective 3D neural network models with limited hardware resources.

Collected by LiDAR sensors, 3D data usually comes in the format of point clouds. Conventionally, researchers rasterize the point cloud into voxel grids and process them using 3D volumetric convolutions [1]. With low resolutions, there will be information loss during voxelization: multiple points will be merged together if they lie in the same grid. Therefore, a high-resolution representation is needed in order to preserve the fine details in the input data. However, the computational cost and memory requirement both increase *cubically* with voxel resolution. Thus, it is infeasible to train a voxel-based model with high-resolution inputs: *e.g.*, 3D-UNet [2] requires more than 10 GB of GPU memory on 64×64×64 inputs with batch size of 16, and the large memory footprint makes it

rather difficult to scale beyond this resolution.

Recently, another stream of models attempt to directly process the 3D point clouds [3], [4], [5], [6]. These point-based models require much lower GPU memory than voxel-based models thanks to the sparse data representation. However, they neglect the fact that the *random memory access* is also very inefficient. As the input points are scattered over the entire 3D space in a very irregular manner, processing them introduces many random memory accesses. Most point-based models [6] mimic the 3D volumetric convolution: *i.e.*, they compute the feature of each point by aggregating its neighboring features. However, neighbors are not stored contiguously in the point representation; therefore, indexing them requires the costly nearest neighbor search. To trade space for time, previous methods replicate the entire point cloud for each center point in the nearest neighbor search, and the memory cost will be $\mathcal{O}(n^2)$, where n is the number of input points. Another very large overhead is introduced by the dynamic kernel computation. Since the relative positions of neighbors are not fixed, these point-based models have to generate the convolution kernels dynamically based on different offsets.

Designing efficient 3D neural networks needs to take the hardware into consideration. Compared with arithmetic operations, memory operations are much more expensive: *i.e.*, they consume two orders of magnitude *higher* energy and have two orders of magnitude *lower* bandwidth (Fig. 1a). Another very important aspect is the memory access pattern: *i.e.*, the random access will introduce memory bank conflicts and decrease the throughput (Fig. 1b). From the hardware perspective, conventional 3D models are inefficient due to large memory footprint and random memory access.

This paper studies 3D deep learning from the perspective of hardware efficiency. After analyzing the bottlenecks of previous methods, we introduce a novel hardware-efficient 3D primitive, *Point-Voxel Convolution (PVConv)*, that brings the

---

- *Z. Liu, H. Tang, S. Zhao, K. Shao, and S. Han are with Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. The first two authors contributed equally to this work.*
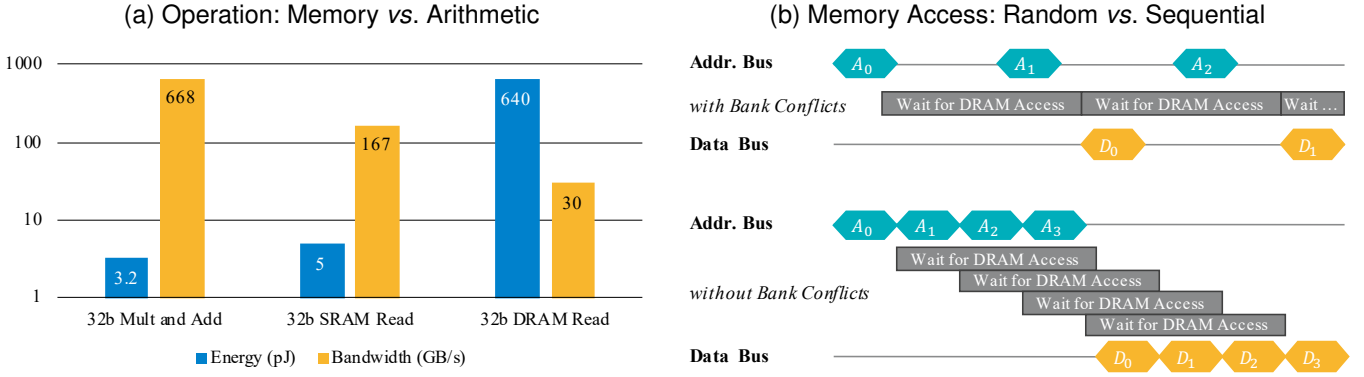 *E-mails:* {zhijian,kentang,shengyuz,kshao23,songhan}@mit.edu.

Fig. 1. Efficient 3D models should reduce memory footprint and avoid random memory accesses. **(a)** Off-chip DRAM accesses take two orders of magnitude more energy than arithmetic operations (640pJ *vs.* 3pJ), whereas the bandwidth is two orders of magnitude lower (30GB/s *vs.* 668GB/s). Efficient 3D model should **reduce the memory footprint**, which is the bottleneck of *voxel-based* methods. **(b)** Random memory access is inefficient since it cannot take advantage of the DRAM burst and will cause bank conflicts, whereas contiguous memory access does not suffer from the above issue. Efficient 3D model should **avoid random memory accesses**, which is the bottleneck of *point-based* methods.

best from previous point-based and voxel-based models. It is composed of a fine-grained point-based branch that keeps the 3D data in high resolution without large memory footprint, and a coarse-grained voxel-based branch which aggregates the neighboring features without random memory accesses. However, as the resolution of its voxel-based branch is still constrained by the memory, this primitive is not effective in processing large scenes. To this end, we propose *Sparse Point-Voxel Convolution (SPVConv)* that enhances our PVConv with the sparse convolution to enable higher resolutions in the voxel-based branch. Based on these efficient 3D primitives, we then propose *3D Neural Architecture Search (3D-NAS)* to automatically explore the optimal 3D network architecture given a resource constraint.

As 3D deep learning has been used in various real-world scenarios (*e.g.*, indoor scenes for AR/VR, and outdoor scenes for autonomous driving), we demonstrate the effectiveness of our proposed method on extensive benchmarks including 3D part segmentation (for objects), 3D semantic segmentation (for indoor and outdoor scenes) as well as 3D object detection (for outdoor scenes). Across all these datasets, our method consistently achieves the state-of-the-art performance with **1.8-23.7×** measured speedup. Furthermore, our method has been deployed into the autonomous racing vehicle of MIT Driverless, achieving larger detection range, lower latency and higher accuracy. We hope that our research can bring inspirations to further explorations in this direction.

2 RELATED WORK

2.1 3D Neural Networks

Conventionally, researchers relied on the volumetric representation and applied the convolution to process 3D data [7], [8], [9], [10], [11], [12]. To name a few, Maturana *et al.* [9] proposed the vanilla volumetric CNN; Qi *et al.* [10] extended 2D CNNs to 3D and systematically analyzed the relationship between 3D CNNs and multi-view CNNs; Wang *et al.* [13] incorporated the octree into the volumetric CNN to reduce the memory consumption. Recent studies suggest that the volumetric representation can be used in 3D shape segmentation [11], [14], [15] and 3D object detection [12] as well.

Due to the sparse nature of 3D data, the dense volumetric representation is inherently inefficient and also inevitably introduces information loss.

PointNet [3] takes advantage of the symmetric function to directly process the unordered point sets in 3D. Later research [4], [5], [16] proposed to stack PointNets hierarchically to model neighborhood information and increase model capacity. Instead of stacking PointNets as basic blocks, another type of methods [6], [17], [18] abstract away the symmetric function using dynamically generated convolution kernels or learned neighborhood permutation function. Some other research, such as SPLATNet [19] that naturally extends the 2D image SPLAT to 3D, and SONet [20] which uses the self-organization mechanism with the theoretical guarantee of invariance to point order, also show great potential in general-purpose 3D modeling with point clouds as input. Apart from these general-purpose models, there are also some attempts for specific 3D tasks. SegCloud [21], SGPN [22], SPGraph [23], ParamConv [24], SSCN [25] and RSNet [26] are specialized for 3D semantic and instance segmentation. As for 3D object detection, F-PointNet [27] is based on the RGB detector and point-based regional proposal networks; PointRCNN [28] follows the similar idea while removing the RGB detector.

Recently, researchers started to pay attention to the efficiency of 3D models. Hu *et al.* [29] proposed to aggressively downsample the point cloud to reduce the computation cost. Riegler *et al.* [1], Wang *et al.* [13], [30] and later Lei *et al.* [31] proposed to reduce the memory of volumetric representation using octrees where areas with lower density occupy fewer voxel grids. Graham *et al.* [25] and Choy *et al.* [32] proposed the sparse convolution to accelerate the vanilla volumetric convolution by keeping the activation sparse and skipping the computations in the inactive regions. However, all these methods still require considerable random memory accesses, which are very inefficient.

2.2 Hardware-Efficient Deep Learning

Extensive attention has been paid to hardware-efficient deep learning for real-world applications. For instance, researchers have proposed to reduce the memory access cost by pruning and quantizing the models [33], [34], [35], [36], [37] or directly
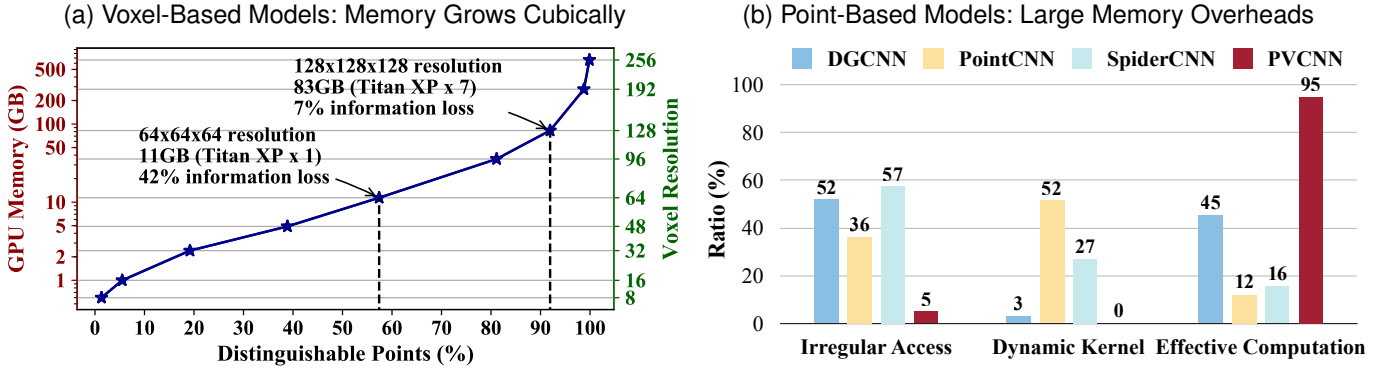
Fig. 2. Both conventional voxel-based and point-based models are inefficient. **(a)** Voxel-based models suffer from the large information loss at acceptable GPU memory consumption. **(b)** Point-based model suffer from large irregular memory access and dynamic kernel computation overheads.

designing the compact models [38], [39], [40], [41], [42], [43]. These approaches are general-purpose and suitable for any neural networks. In this paper, we instead accelerate 3D neural networks based on some domain-specific properties: *e.g.*, 3D point clouds are highly sparse and spatially structured.

2.3 Neural Architecture Search

To alleviate the burden of manually designing neural networks [38], [39], [40], [42], [43], researchers have introduced neural architecture search (NAS) to automatically architect the neural network with high accuracy using reinforcement learning [44], [45] and evolutionary search [46]. A new wave of research started to design efficient models with neural architecture search [47], [48], [49], [50] for edge deployment. However, conventional frameworks require high computation cost and considerable carbon footprint [51]. In order to tackle these, researchers have introduced different techniques to reduce the search cost, including differentiable architecture search [52], path-level binarization [53], single-path one-shot sampling [54], [55], [56], and weight sharing [50], [56], [57]. Furthermore, neural architecture search has also been used in compressing and accelerating neural networks, including pruning [35], [58], [59], [60], [61] and quantization [37], [54], [62], [63]. Most of these methods are tailored for 2D visual recognition, which has many well-defined search spaces [64]. Lately, researchers have applied neural architecture search to 3D medical image segmentation [65], [66], [67], [68], [69], [70] as well as 3D shape classification [71], [72]. However, they are not directly applicable to 3D scene understanding since 3D medical data are still in the similar format as 2D images (which are entirely different from 3D scenes), and 3D objects are of much smaller scales than 3D scenes (which makes them less sensitive to the resolution).

3 ANALYSIS OF EFFICIENCY BOTTLENECKS

3D data usually comes in the format of point clouds:

$$x = \{x_k\} = \{(p_k, f_k)\}, \tag{1}$$

where p_k is the coordinate of the kth point, and f_k is the feature corresponding to p_k. The voxelized representation can also be unified into this formulation, where p_k stands

for the coordinate of the kth voxel grid. Based on this, voxel-based and point-based convolution can be formulated as

$$y_k = \sum_{x_i \in \mathcal{N}(x_k)} \mathcal{K}(x_k, x_i) \times \mathcal{F}(x_i). \tag{2}$$

During the convolution, we iterate x_k over the entire input. For each center x_k, we first index its neighbors x_i in $\mathcal{N}(x_k)$, then convolve the neighboring features $\mathcal{F}(x_i)$ with the kernel $\mathcal{K}(x_k, x_i)$, and produces the corresponding output y_k.

3.1 Voxel-Based Models: Large Memory Footprint

Conventionally, researchers rasterize the point cloud into voxel grids and process them with 3D volumetric convolutions [1]. Voxel-based representation is regular and has good memory locality. However, it requires very high resolution in order not to lose much information. When the resolution is low, multiple points are bucketed into the same voxel grid, and these points will no longer be *distinguishable*. A point is kept only when it exclusively occupies one voxel grid. In Fig. 2a, we investigate the number of distinguishable points and the memory consumption (during training with batch size of 16) with different resolutions. On a single GPU (with 12 GB of memory), the largest affordable resolution is 64, which will lead to **42%** of information loss. To keep more than 90% of the information, we then need to double the resolution to 128, consuming 7.2× GPU memory (**82.6 GB**), which is prohibitive in constrained scenarios. Although the GPU memory increases cubically with the resolution, the number of distinguishable points has a diminishing return. Therefore, the voxel-based solution is not scalable.

3.2 Point-Based Models: Sparse Data Organization

Recently, another stream of models process the point cloud directly [3], [4], [6], [16], [18]. These point-based models require much lower GPU memory than voxel-based models thanks to the sparse representation. Among them, PointNet [3] is also computation efficient, but it lacks the local context modeling capability. All the other models [4], [6], [16], [18] improve the expressiveness of PointNet by aggregating the neighborhood information in the point-based domain. However, this will lead to the irregular memory access pattern and introduce the dynamic kernel computation overhead, which becomes the efficiency bottleneck.

(a) Voxel-Based Feature Aggregation (*Coarse-Grained*)

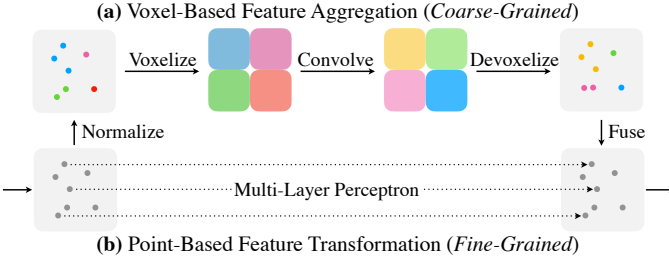**(b)** Point-Based Feature Transformation (*Fine-Grained*)

Fig. 3. Point-Voxel Convolution (PVConv) is composed of a *low-resolution* voxel-based branch and a *high-resolution* point-based branch. The voxel-based branch extracts *coarse-grained* neighborhood information, which is supplemented by *fine-grained* individual point features extracted from the point-based branch.

Irregular Memory Access. Different from the voxel-based representation, neighboring points $x_i \in \mathcal{N}(x_k)$ in the point-based representation will not be laid out contiguously in the memory. Besides, 3D points are scattered in $\mathbb{R}^3$; therefore, we need to explicitly identify who are in the neighboring set $\mathcal{N}(x_k)$, rather than by direct indexing. Point-based methods often define $\mathcal{N}(x_k)$ as nearest neighbors in the coordinate space [6], [18] or the feature space [16]. Either requires explicit and expensive KNN computation. After KNN, gathering all neighbors x_i in $\mathcal{N}(x_k)$ will require large amount of random memory accesses, which is not cache friendly. Combining the cost of neighbor indexing and data movement, the state-of-the-art point-based models spend **36**% [6], **52**% [16] and **57**% [18] of the total runtime on structuring the irregular data and accessing the memory randomly (Fig. 2b).

Dynamic Kernel Computation. For 3D volumetric convolutions, the kernel value $\mathcal{K}(x_k, x_i)$ can be directly indexed because the relative positions of the neighbor x_i are fixed for different center x_k: *e.g.*, each axis of the offset $p_i - p_k$ can only be 0, ± 1 for the convolution with size of 3. For the point-based convolution, the points are scattered over the entire 3D space irregularly; therefore, the relative positions of neighbors become unpredictable. In this case, we will have to calculate the kernel $\mathcal{K}(x_k, x_i)$ for each neighbor x_i *on the fly*. For instance, SpiderCNN [18] leverages the third-order Taylor expansion as a continuous approximation of the kernel $\mathcal{K}(x_k, x_i)$; PointCNN [6] permutes the neighboring points into a canonical order with the feature transformer $\mathcal{F}(x_i)$. Both will introduce additional matrix multiplications. Empirically, the overhead of dynamic kernel computation for PointCNN can be more than **50**% (Fig. 2b).

The combined overhead of irregular memory access and dynamic kernel computation ranges from **55**% (DGCNN) to **88**% (PointCNN). Thus, most computations are wasted on dealing with the irregularity of point-based representation.

4 DESIGNING EFFICIENT 3D PRIMITIVES

Based on our analysis of efficiency bottlenecks, we introduce a novel hardware-efficient 3D primitive for small 3D objects as well as indoor scenes: *Point-Voxel Convolution (PVConv)*. It combines the advantages of point-based methods (small memory footprint) and voxel-based methods (good data locality and regularity). For large outdoor scenes, we further propose *Sparse Point-Voxel Convolution (SPVConv)* that en-

hances PVConv with the sparse convolution to enable higher resolutions in the voxel-based branch.

4.1 Point-Voxel Convolution (PVConv)

Our PVConv disentangles the *fine-grained* feature transformation and the *coarse-grained* neighbor aggregation so that each branch can be implemented very efficiently and effectively. As in Fig. 3, the upper voxel-based branch first transforms the points into *low-resolution* voxel grids, then it aggregates the neighboring points with voxel-based convolutions, followed by the devoxelization to convert them back to points. Either voxelization or devoxelization requires a single scan over all points, making the memory cost low. The lower point-based branch extracts the features for each individual point. As it does not aggregate the neighbor's information, it is able to afford a very *high resolution*.

4.1.1 Voxel-Based Feature Aggregation

A key component of convolution is to aggregate the neighboring information in order to extract the local features. We choose to perform this feature aggregation in the volumetric domain due to its regularity.

Normalization. The scale of different point clouds might be different. Thus, we normalize the coordinates $\{p_k\}$ before converting the point cloud into the volumetric domain. First, we translate all points into the local coordinate system with the gravity center as the origin. After that, we normalize the points into the unit sphere by dividing all coordinates by $\max \|p_k\|_2$, and then scale and translate the points to $[0, 1]$. We denote the normalized coordinates as $\{\hat{p}_k\}$.

Voxelization. We transform the normalized point cloud $\{(\hat{p}_k, f_k)\}$ into the dense voxelized representation $\{V_{u,v,w}\}$ by averaging all of the features f_k whose coordinate $\hat{p}_k = (\hat{x}_k, \hat{y}_k, \hat{z}_k)$ falls into the voxel grid (u, v, w):

$$V_{u,v,w,c} = \sum_{k=1}^{n} \mathbb{I}[\lfloor \hat{x}_k r \rfloor = u, \lfloor \hat{y}_k r \rfloor = v, \lfloor \hat{z}_k r \rfloor = w] \times f_{k,c}/N_{u,v,w}, \tag{3}$$

where r denotes the voxel resolution, $\mathbb{I}[\cdot]$ is the binary indicator of whether the coordinate $\hat{p}_k$ belongs to the voxel grid (u, v, w), $f_{k,c}$ denotes the cth channel feature corresponding to $\hat{p}_k$, and $N_{u,v,w}$ is the normalization factor (*i.e.*, the number of points that fall in that voxel grid).

Feature Aggregation. After converting the points into the voxel grids, we apply a stack of 3D volumetric convolutions to aggregate the features. Similar to conventional 3D models, we apply batch normalization [73] and nonlinear activation function [74] after each 3D volumetric convolution.

Devoxelization. As we need to fuse the information with the point-based feature transformation branch, we transform the voxel-based features back to the domain of point cloud. A simple implementation of the voxel-to-point mapping is the nearest-neighbor interpolation (*i.e.*, assign the feature of a grid to all points in it). However, this implementation will make the points in the same voxel grid always share the same feature values. Therefore, we instead leverage the trilinear interpolation to transform the voxel grids to points to make sure that the features mapped to each point are distinct.

4.1.2 Point-Based Feature Transformation

The voxel-based feature aggregation branch fuses the neighborhood information in a coarse granularity. However, in order to model finer-grained individual point features, low-resolution voxel-based methods alone might not be sufficient. Hence, we directly operate on each point to extract individual point features using an MLP. Though simple, the MLP outputs distinct and discriminative features for each point. Such high-resolution individual point information is very critical to supplement the coarse-grained voxel-based information. With individual point features and aggregated neighborhood information, we can fuse two branches efficiently with an addition as they are providing complementary information.

4.1.3 Analysis

PVConv is much better than previous voxel-based and point-based models in terms of both efficiency and effectiveness.

Better Locality and Regularity. Our PVConv is more efficient than conventional point-based convolutions due to its better data locality and regularity. Our voxelization and devoxelization both require only $\mathcal{O}(n)$ random memory accesses, where n is the number of points, since we only need to iterate over all points once to scatter them to their corresponding grids. However, for conventional point-based methods, gathering neighbors for all points will require at least $\mathcal{O}(kn)$ random memory accesses, where k is the number of neighbors. Thus, our PVConv is $k\times$ more efficient from this viewpoint. As the typical value for k is 32/64 in PointNet++ [4] and 16 in PointCNN [6], PVConv empirically reduces the number of incontiguous memory accesses by 16-64$\times$, achieving better data locality. Besides, as our convolutions are done in the voxel domain, which is regular, our PVConv does not require KNN computation and dynamic kernel computation, which are usually quite expensive.

Higher Resolution. As our point-based feature extraction branch is implemented as MLP, a natural advantage is that we are able to maintain the same number of points throughout the whole network while still having the capability to model neighborhood information. Let us make a comparison between our PVConv and the set abstraction (SA) module in PointNet++ [4]. Suppose we have a batch of 2048 points with 64-channel features (with batch size of 16), and we then aggregate information from 125 neighbors of each point and transform the aggregated feature to output the features with the same size. In this case, the SA module requires 75.2 ms of latency and 3.6 GB of memory, while our PVConv only requires 25.7 ms of measured latency and 1.0 GB of memory. The SA module will have to downsample to 685 points (*i.e.*, around 3$\times$ downsampling) to match up with the latency of our PVConv, while the memory consumption will still be 1.5$\times$ higher. Therefore, with the same latency, our PVConv is capable of modeling the full point cloud, while the SA module has to downsample the input aggressively, which will inevitably induce information loss.

4.2 Sparse Point-Voxel Convolution (SPVConv)

PVConv is very efficient and effective especially for small 3D objects and indoor scenes as their scales are fairly small; however, it is less suitable for large outdoor scenes due to the coarse voxelization. PVConv-based model can afford the

TABLE 1
Comparison between PVConv and SPVConv in Large Outdoor Scenes

	Input	Voxel Size (m)	Latency (ms)	Mean IoU
PVConv	Sliding Window	0.05	35640	–
	Entire Scene	0.80	146	39.0
SPVConv	Entire Scene	0.05	**85**	**58.8**

PVConv is not suitable for large scenes. If processing with sliding windows, its latency is not affordable for deployment. If taking the whole scene, its resolution is too coarse to capture useful information.

resolution of at most 128 in its voxel-based branch on a single GPU (with 12 GB of memory). For a large outdoor scene (with size of 100m×100m×10m), each voxel grid will correspond to a large area (with size of 0.8m×0.8m×0.1m). In this case, small instances (*e.g.*, pedestrians) will only occupy very few voxel grids. From such few points, PVConv can hardly learn any useful information from the voxel-based branch, leading to a relatively low performance (see Table 1). Alternatively, we can process the large 3D scenes piece by piece so that each sliding window is of smaller scale. In order to preserve the fine-grained information, we found empirically that the voxel size needs to be lower than 0.05m. In this case, we have to run the model once for each of the 244 sliding windows, which will take 35 seconds to process a single scene. Such a large latency is not affordable for most real-time applications (*e.g.*, autonomous driving).

To this end, we introduce *Sparse Point-Voxel Convolution (SPVConv)* to effectively and efficiently process the large 3D scene. It follows a similar two-branch architectural design as PVConv except that the volumetric convolution in the voxel-based branch is replaced with the sparse convolution [32]. As point clouds are intrinsically very sparse, this modification can significantly reduce the memory consumption if we use a higher resolution in the voxel-based branch. Furthermore, SPVConv can also be considered as adding a high-resolution point-based branch to the vanilla sparse convolution so that the fine details (*i.e.*, small instances) can be preserved.

Most of the operations in PVConv can be directly adapted to the sparse voxelized representation. However, the voxelization and devoxelization is not as trivial since we cannot easily index a given 3D coordinate in the sparse voxelized representation. A straightforward implementation based on the iterative coordinate comparison will require $\mathcal{O}(mn)$ of time, where m is the number of points in the point cloud representation, and n is the number of activated points in the sparse voxelized representation. As m and n are typically at the order of 10^5, this naive implementation is not practical for real-time applications. Instead, we propose to use the parallel hash tables on GPUs to accelerate the sparse voxelization and devoxelization. Note that existing implementations [25], [32] use CPU-based hash tables for kernel map construction in the sparse convolution, which is much slower. Concretely, we first construct a hash table for all activated points in the sparse voxelized representation, which can be finished in $\mathcal{O}(n)$ of time. After that, we iterate over all points, and for each point, we then query its coordinate in the hash table to obtain its corresponding index in the sparse voxelized representation. As the lookup over the hash table requires
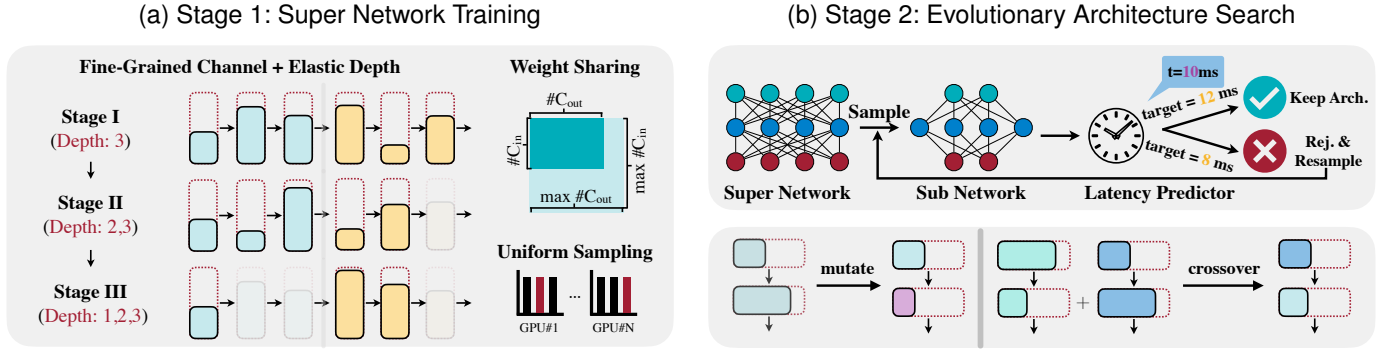
(a) Stage 1: Super Network Training   (b) Stage 2: Evolutionary Architecture Search

Fig. 4. We propose a two-stage 3D Neural Architecture Search (3D-NAS) framework to automatically design efficient 3D deep learning architectures. **(a)** In the first stage, we train a *super network* that supports all candidate networks within the design space. **(b)** In the second stage, we perform *evolutionary architecture search* to find the best candidate network given a specific resource constraint.

$\mathcal{O}(1)$ time in the worst case, this query step will in total take $\mathcal{O}(m)$ time. Therefore, the total time of coordinate indexing will be reduced from $\mathcal{O}(mn)$ to $\mathcal{O}(m+n)$.

5 SEARCHING EFFICIENT 3D ARCHITECTURES

Even with the efficient 3D primitive, designing an efficient neural network model is still challenging. We need to carefully adjust the network architecture (*e.g.*, channel numbers and kernel sizes of all layers) to meet the requirements for real-world applications (*e.g.*, latency, energy, and accuracy). In this section, we propose *3D Neural Architecture Search (3D-NAS)* to automatically design efficient 3D models (Fig. 4). We first carefully design a search space tailored for 3D and then introduce our training paradigm that supports a large number of neural networks within a single super network. Finally, we use the evolutionary search to explore the best candidate in the design space given a resource constraint.

5.1 Design Space

The performance of neural architecture search is impacted by the design space quality. In our search space, we incorporate fine-grained channel numbers and elastic network depths; however, we do not support different kernel sizes.

5.1.1 Primitive Selection

PVConv accesses the memory contiguously and has fairly small memory footprint for small objects and indoor scans. However, it does not scale up very well to large-scale outdoor scenes (Table 1) as the resolution of its voxel-based branch is still constrained by the memory. SPVConv, on the other hand, can efficiently scale up to large scans but falls short in modeling small objects and regions due to the irregular overhead introduced by sparse operations. As in Fig. 5, the latency of SPVConv almost stays constant when the voxel resolution increases from 4 to 48, but the latency of PVConv quickly grows when the resolution scales up. As a result, at smaller resolutions, PVConv can be up to $3.6\times$ faster than SPVConv, while SPVConv becomes $3.3\times$ faster than PVConv once the resolution is larger than 48. As the spatial range of single objects/indoor scans is smaller than 1.5m×1.5m×1.5m, it is sufficient to use voxel resolution smaller than 32, and therefore, PVConv is favored over SPVConv. On the other hand, SPVConv is favored when the spatial range scales far beyond 1.5m×1.5m×1.5m. We will validate this in Section 6.
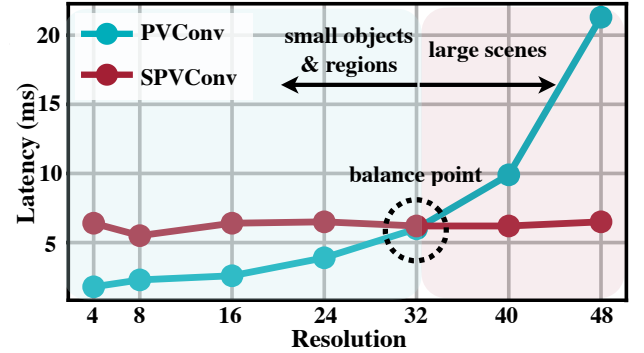

Fig. 5. PVConv is more efficient and effective at smaller resolutions while SPVConv is more efficient at larger resolutions. Here, the GPU latency is measured on NVIDIA GTX 1080 Ti.

5.1.2 Fine-Grained Channel Numbers

The computation cost increases quadratically with the number of channels; therefore, the channel number selection has a large influence on the network efficiency. Most existing neural architecture frameworks [53] only support the coarse-grained channel number selection: *e.g.*, searching the expansion ratio of the ResNet/MobileNet blocks over a few (2-3) choices. In this case, only intermediate channel numbers of the blocks can be changed; while the input and output channel numbers will still remain the same. Empirically, we observe that this limits the variety of the search space. To this end, we enlarge the search space by allowing all channel numbers to be selected from a large collection of choices (with size of $O(n)$). This fine-grained channel number selection largely increases the number of candidates for each block: *e.g.*, from 2-3 to $\mathcal{O}(n^2)$ for a block with two convolutions.

5.1.3 Elastic Network Depth

For 3D CNNs, reducing the channel numbers alone cannot achieve significant measured speedup, which is different from normal 2D CNNs. For example, by shrinking all channel numbers in MinkowskiNet by $4\times$ and $8\times$, the number of MACs is reduced to 7.5 G and 1.9 G, respectively. However, although the number of MACs is drastically reduced, their measured latency on the GPU is very similar: 105 ms and 96 ms (on NVIDIA GTX 1080 Ti GPU). This suggests that scaling

down the number of channels alone is not able to offer us with very efficient models, even though the number of MACs is very small. This might be because 3D modules are usually more memory-bounded than 2D modules; the number of MACs decreases quadratically with channel number, while memory decreases linearly. Thus, we incorporate the elastic network depth into our design space so that layers with very small computation (and large memory cost) can be removed and merged into their neighboring layers.

5.1.4 Small Kernel Matters

Kernel sizes are usually included into the search space of 2D CNNs. This is because a single convolution with larger kernel size can be more efficient than multiple convolutions with smaller kernel sizes on GPUs. However, it is not the case for 3D CNNs. From the perspective of computation cost, a single 2D convolution with kernel size of 5 requires only $1.4\times$ more MACs than two 2D convolutions with kernel sizes of 3; while a single 3D convolution with kernel size of 5 requires $2.3\times$ more MACs than two 3D convolutions with kernel sizes of 3 (if applied to dense voxel grids). This larger computation cost makes it less suitable to use large kernel sizes in 3D CNNs. Furthermore, the computation overhead of 3D modules is also related to the kernel sizes. For example, the sparse convolution requires $\mathcal{O}(k^3 n)$ time to build the kernel map, where k is the kernel size and n is the number of points, which indicates that its cost grows cubically with respect to the kernel size. Based on these reasons, we decide to keep the kernel size of all convolutions to be 3 and do not allow the kernel size to change in our search space. Even with the small kernel size, we can still achieve a large receptive field by changing the network depth, which can achieve the same effect as changing the kernel size.

5.2 Training Paradigm

Searching over a fine-grained design space is very challenging since it is impossible to train every sampled candidate network randomly from scratch [47]. Motivated by Guo *et al.* [54], we incorporate all candidate networks into a single super network, and after training this super network once, each candidate network can then be extracted with inherited weights. The total training cost during the neural architecture search can then be reduced from $\mathcal{O}(n)$ to $\mathcal{O}(1)$, where n is the number of candidate networks.

5.2.1 Uniform Sampling

At each training iteration, we randomly sample a candidate network from the super network: *i.e.*, randomly select the channel number for each layer, and then randomly select the network depth (*i.e.* the number of blocks to be used) for each stage. The total number of candidate networks to be sampled during training is very limited; thus, we choose to sample different candidate networks on different GPUs and average their gradients at each step so that more candidate networks can be sampled. For 3D, this is more critical because the 3D datasets usually contain fewer samples than the 2D datasets: *e.g.*, 20K on SemanticKITTI [75] *vs.* 1M on ImageNet [76].

5.2.2 Weight Sharing

As the number of candidate networks is enormous, every candidate network will only be optimized for a small fraction of the total schedule. Therefore, uniform sampling alone is not enough to train all candidate networks sufficiently (*i.e.*, achieving the same level of performance as being trained from scratch). To tackle this, we adopt the weight sharing technique so that every candidate network can be optimized at each iteration even if it is not sampled. Specifically, given the input channel number C_{in} and output channel number C_{out} of each convolution layer, we simply index the first C_{in} and C_{out} channels from the weight tensor accordingly to perform the convolution [54]. For each batch normalization layer, we similarly crop the first c channels from the weight tensor based on the sampled channel number c. Finally, with the sampled depth d for each stage, we choose to keep the first d layers, instead of randomly sampling d of them. This ensures that each layer will always correspond to the same depth index within the stage.

5.2.3 Progressive Depth Shrinking

Suppose we have n stages, each of which has m different depth choices ranging from 1 to m. If we sample the depth d_k for each stage k randomly, the expected total depth of the network will be $\mathbb{E}[d] = \sum_{k=1}^{n} \mathbb{E}[d_k] = n(m+1)/2$, which is much smaller than the maximum depth nm. Furthermore, the probability of the largest candidate network (with the maximum depth) being sampled is extremely small: m^{-n}. Therefore, the largest candidate networks are poorly trained due to the small possibility of being sampled. To this end, we introduce progressively depth shrinking to alleviate this issue. We divide the training epochs into m segments for m different depth choices. During the kth training segment, we only allow the depth of each stage to be selected from $m - k + 1$ to m. This is essentially designed to enlarge the search space gradually so that these large candidate networks can be sampled more frequently.

5.3 Search Algorithm

After the super network is fully trained, we use the evolutionary architecture search to find the best network architecture under a certain resource constraint.

5.3.1 Resource Constraint

We support #MACs and measured latency as our resource constraint for candidate networks.

#MACs Constraint. #MACs is an implementation- and hardware-agnostic efficiency metric. Unlike dense 2D CNNs, #MACs of sparse 3D CNNs cannot be simply determined by the input size and network architecture. This is because the sparse convolution only performs the computation over the active synapses; thus, its computation cost is also related to the size of kernel map, which is determined by the input sparsity pattern. To address this, we first estimate the average kernel map size over the entire dataset for each convolution layer and then sum them up to compute the average #MACs.

Latency Constraint. We support to directly use the measured latency on the target hardware as our resource constraint as well. We first encode each candidate network into a 1D architecture vector. We then randomly sample 50,000

TABLE 2
Results of Object Part Segmentation on ShapeNet Part

	#Par. (M)	#MACs (G)	Mem. (G)	Lat. (ms)	mIoU
PointNet [3]	2.5	5.3	1.5	15.1	83.7
3D-UNet [2]	8.1	2996.9	8.8	682.1	84.6
RSNet [26]	6.9	1.4	0.8	74.6	84.9
PointNet++ [4]	1.8	4.9	2.0	77.9	85.1
DGCNN [16]	1.5	18.5	2.4	87.8	85.1
SPVCNN (0.25×C)	0.3	0.3	1.1	20.2	84.4
PVCNN (0.25×C)	**0.3**	1.0	**0.8**	8.3	**85.2**
SPVNAS	1.7	1.0	1.2	18.4	85.2
PVNAS-A	0.4	**0.9**	0.9	**7.0**	**85.2**
SpiderCNN [18]	2.6	10.6	6.5	170.7	85.3
SPVCNN (0.5×C)	1.1	1.3	1.4	24.7	85.1
PVCNN (0.5×C)	1.1	3.9	**1.0**	16.0	85.5
PVNAS-B	**0.6**	**2.2**	**1.0**	**11.6**	85.5
PVNAS-C	0.7	2.9	**1.0**	14.0	**85.6**
PointConv [77]	21.6	**11.6**	6.5	163.7	85.7
PointCNN [6]	8.3	26.9	2.5	135.8	86.1
SPVCNN (1×C)	4.2	5.3	2.0	38.0	85.6
PVCNN (1×C)	**4.2**	15.3	**1.6**	41.4	86.2

*On average, PVCNN outperforms point-based models with **5.5×** speedup and **3×** memory reduction, and outperforms the voxel-based baseline with **59×** measured speedup and **11×** memory reduction.*

candidate networks from the design space and measure their latency on the target hardware. With the collected pairs of architecture vector and measured latency, we finally train an MLP regressor to estimate the latency based on the network architecture. The resulting predictor can accurately predict the latency of different candidate networks with a relative error of less than 2% on both edge and cloud GPUs.

5.3.2   Evolutionary Search

We automate the architecture search with the evolutionary algorithm [54]. We initialize the starting population with n randomly sampled candidate networks. At every iteration, we evaluate all candidate networks in the population and select the k models with the highest accuracy (*i.e.*, the fittest individuals). The population for the next iteration is then generated with $(n/2)$ mutations and $(n/2)$ crossovers. For each mutation, we randomly select one among the top-k candidates and modify each of its architectural parameters (*e.g.*, channel numbers, network depths) with a pre-defined probability; for each crossover, we select two from the top-k candidates and produce a new network by fusing them together randomly. Finally, the best network architecture is obtained from the population of the last iteration. During the evolutionary search, we ensure that all candidate networks in the population always meet the given resource constraint (we will resample another candidate network until the resource constraint is satisfied).

6   EXPERIMENTS

In this section, we evaluate our models on six representative 3D benchmark datasets:

- ShapeNet [8] (coarse-grained object part segmentation),
- PartNet [78] (fine-grained object part segmentation),

- S3DIS [79], [80] (indoor scene segmentation),
- SemanticKITTI [75] (outdoor scene segmentation),
- nuScenes [81] (outdoor scene segmentation),
- KITTI [82] (outdoor object detection).

On these datasets, we evaluate four variants of our method:

- **PVCNN** (manually-designed model with PVConv),
- **PVNAS** (3D-NAS applied to PVCNN),
- **SPVCNN** (manually-designed model with SPVConv),
- **SPVNAS** (3D-NAS applied to SPVCNN).

6.1   3D Object Part Segmentation

6.1.1   ShapeNet

We first evaluate our method on 3D part segmentation and conduct experiments on the large-scale 3D object dataset, ShapeNet [8]. As 3D objects are very small, it is suitable to process them using PVConv. Therefore, we build our PVCNN by replacing the MLP layers in PointNet [3] with PVConv's.

Baselines. We use PointNet [3], RSNet [26], PointNet++ [4] (with multi-scale grouping), DGCNN [16], SpiderCNN [18] and PointCNN [6] as point-based baselines. We reimplement 3D-UNet [2] as voxel-based baseline. For a fair comparison, we follow the same evaluation protocol as in Li *et al.* [6] and Graham *et al.* [25]. The evaluation metric is mean intersection-over-union (mIoU): we first calculate the part-averaged IoU for each of the 2,874 test models and average the values as the final metrics. Besides, we report the measured GPU latency and GPU memory consumption on a single NVIDIA GTX 1080 Ti GPU. We ensure the input data to have the same size with 2048 points and batch size of 8.

Results. As in Table 2, PVCNN outperforms all previous models. It directly improves the accuracy of its backbone (PointNet) by 2.5% with even smaller overhead compared with PointNet++. Besides, we also design narrower versions of our PVCNN by reducing the number of channels to 25% (0.25×C) and 50% (0.5×C). The resulting model requires only 46.4% latency of PointNet, and it still outperforms several point-based methods with sophisticated neighborhood aggregation including RSNet, PointNet++ and DGCNN, which are almost an order of magnitude slower. PVCNN achieves a much better accuracy *vs.* latency trade-off compared with all point-based methods. With similar accuracy, PVCNN is **24×** faster than SpiderCNN and **3.3×** faster than PointCNN. Our PVCNN also achieves a significantly better accuracy *vs.* memory trade-off compared with the voxel-based baseline. With better accuracy, PVCNN can save the GPU memory consumption by **10×** compared with 3D-UNet. We further apply 3D-NAS to PVCNN under different latency constraints to obtain a family of PVNAS models. With the same accuracy as PVCNN (0.25×C), PVNAS-A achieves 1.2× faster inference speed. It also outperforms PointNet by 1.5 mIoU with **3×** measured speedup. In comparison with PVCNN (0.5×C), PVNAS-B achieves 1.4× speedup with no loss of accuracy, and PVNAS-C achieves 1.1× speedup with better mIoU.

PV/SPVConv. On ShapeNet, PVCNN/PVNAS achieves far better efficiency-accuracy trade-offs compared with SPVC-NN/SPVNAS. Specifically, PVCNN achieves similar or better mIoU than SPVCNN with **2.4×** measured speedup. Similarly, PVNAS-A achieves the same accuracy as SPVNAS with **2.6×** lower latency. These results validate our claim in Section 5.1.1 that PVConv is more favorable for modeling small 3D objects.

(a) Top Row: Features Extracted from *Coarse-Grained* Voxel-Based Branch (Large, Continuous)

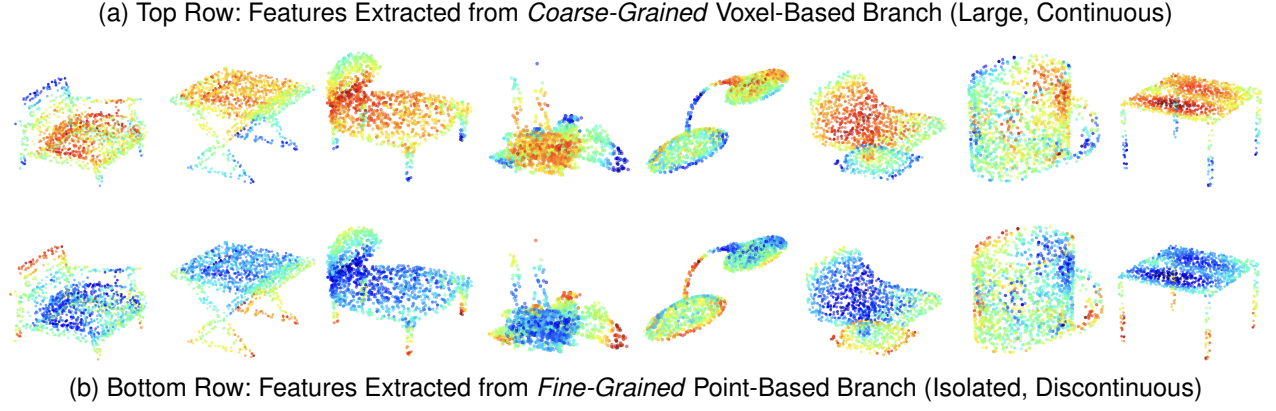(b) Bottom Row: Features Extracted from *Fine-Grained* Point-Based Branch (Isolated, Discontinuous)

Fig. 6. Two branches are providing complementary information: the voxel-based branch focuses on the large, continuous parts, while the point-based focuses on the isolated, discontinuous parts.

TABLE 3
Results of Fine-Grained Object Part Segmentation on PartNet

	#P (M)	#M (G)	L (ms)	mIoU	Bed	Bott	Chair	Clock	Dish	Disp	Door	Ear	Fauc	Knife	Lamp	Micro	Frid	Stora	Table	Trash	Vase
PointNet [3]	2.5	25.1	9.4	35.6	13.4	29.5	27.8	28.4	48.9	76.5	30.4	33.4	47.6	32.9	18.9	37.2	33.5	38.0	29.0	34.8	44.4
PointNet++ [4]	1.8	5.4	26.0	42.5	30.3	41.4	39.2	41.6	50.1	80.7	32.6	38.4	52.4	34.1	25.3	48.5	36.4	40.5	33.9	46.7	49.8
Deep LPN [83]	2.7	3.0	206.8	38.6	29.5	42.1	41.8	34.7	33.2	81.6	34.8	49.6	53.0	44.8	28.4	33.5	32.3	41.1	36.3	43.1	57.8
SpiderCNN [18]	2.6	52.2	2170.8	37.0	36.2	32.2	30.0	24.8	50.0	80.1	30.5	37.2	44.1	22.2	19.6	43.9	39.1	44.6	20.1	42.4	32.4
PointCNN [6]	8.3	71.9	106.6	46.4	41.9	41.8	43.9	36.3	58.7	82.5	37.8	48.9	60.5	34.1	20.1	58.2	42.9	49.4	21.3	53.1	58.9
ResGCN [84]	3.8	55.9	771.3	45.1	35.9	49.3	41.1	33.8	56.2	81.0	31.1	45.8	52.8	44.5	23.1	51.8	34.9	47.2	33.6	50.8	54.2
SPVCNN (0.5×C)	0.3	1.6	13.5	43.6	32.8	45.1	35.3	33.7	58.1	79.5	36.2	49.4	48.7	39.5	22.7	49.3	38.1	41.2	27.2	49.1	55.4
PVCNN (0.5×C)	0.3	2.3	4.4	47.2	35.4	44.5	37.0	38.9	63.2	81.4	44.2	52.1	53.5	46.0	23.1	54.5	43.6	44.1	30.3	52.7	57.3
SPVCNN (0.72×C)	0.6	3.3	15.4	44.3	33.3	47.2	36.9	36.0	59.6	79.5	34.9	49.0	50.4	37.4	23.9	48.9	40.4	42.6	27.8	50.1	54.5
PVCNN (0.72×C)	0.6	4.7	5.5	47.3	35.6	43.5	38.8	39.5	62.9	81.0	42.6	53.0	54.5	42.9	24.4	53.5	42.5	45.6	32.0	52.1	59.7
SPVCNN (1×C)	1.1	6.4	16.9	45.0	32.9	46.1	37.6	33.5	58.7	80.0	38.3	51.6	50.6	44.6	23.4	49.3	38.7	44.1	29.8	51.0	54.7
PVCNN (1×C)	1.1	9.1	6.9	47.8	35.9	43.9	39.8	40.0	61.5	81.6	44.9	52.5	54.6	45.0	26.0	55.2	44.3	46.2	32.7	51.6	56.7
SGAS [85]	–	–	143*	48.3	43.4	50.8	41.2	38.8	61.4	82.6	37.1	48.8	56.1	49.4	21.2	56.5	44.5	49.4	29.3	54.4	56.0
LC-NAS-14 [86]	–	–	152*	48.6	41.9	51.7	39.7	39.6	61.5	82.5	39.3	49.0	54.7	55.3	22.2	55.1	45.2	48.0	30.3	54.6	54.9
LC-NAS-18 [86]	–	–	185*	46.6	40.7	50.5	39.9	39.5	59.8	82.2	35.0	44.5	53.2	44.9	22.0	54.1	41.5	45.8	31.5	53.0	54.4
SPVNAS	0.4	1.2	13.2	46.8	36.7	48.2	39.4	35.7	61.9	81.8	40.0	54.2	55.1	40.4	25.3	50.8	41.3	45.9	31.3	51.5	56.7
PVNAS-A	0.3	2.7	4.2	47.5	35.2	44.1	39.5	36.6	59.6	81.8	44.8	52.4	54.7	47.3	23.2	54.1	45.0	45.8	32.5	53.4	57.6
PVNAS-B	0.4	3.9	4.9	48.0	37.3	44.5	40.2	36.0	60.6	82.1	43.8	52.3	55.8	47.3	23.8	52.2	47.8	47.8	34.1	52.7	57.5

*Here, **#P** denotes the number of parameters, **#M** denotes the number of MACs, and **L** denotes the measured latency (on a single NVIDIA GTX 1080 Ti GPU). *: numbers are from Li et al. [86], which are measured on a single NVIDIA RTX 2080 GPU.*

(a) PVCNN *vs.* PointNet
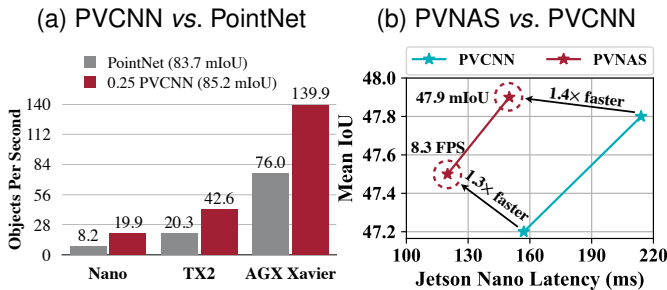

(b) PVNAS *vs.* PVCNN

Fig. 7. PVCNN achieves **real-time** 3D object segmentation with 2,048 input points on edge devices. With PVNAS, we further boost the efficiency on NVIDIA Jetson Nano, achieving **8.3 FPS** with **10,000** input points.

Visualization. We visualize the voxel and point features from the final PVConv, where warmer color indicates larger magnitude. As in Fig. 6, the voxel branch tends to capture large, continuous parts (*e.g.*, table top, lamp head) while the point branch captures isolated, discontinuous details (*e.g.*, table legs, lamp neck). Two branches indeed provide complementary information. This is aligned with our design.

6.1.2 PartNet

We also conduct experiments on the more challenging fine-grained 3D object part segmentation benchmark, PartNet [78]. Different from ShapeNet, where each object is annotated with 2 to 6 parts, objects in PartNet can have as many as 50 parts. To perform fine-grained segmentation, the models usually take in a batch of 10,000 points instead of 2,048 points.

Baselines. We compare PVCNN with state-of-the-art point-based methods including PointNet [3], PointNet++ [4], Deep LPN [83], SpiderCNN [18], PointCNN [6] and ResGCN [84]. We also compare PVNAS with automatically-designed point cloud segmentation networks including SGAS [85] and LC-NAS [86]. To ensure fair comparisons, we follow the original experiment setting in Mo *et al.* [78] to train separate models for different object classes. The results of our method are

TABLE 4
Results of Indoor Scene Segmentation on S3DIS

	#Par. (M)	#MACs (G)	Mem. (GB)	Lat. (ms)	mIoU
PointNet [3]	1.2	3.6	1.0	12.1	43.0
PVCNN (0.125×C)	0.04	0.3	**0.6**	**6.2**	**46.9**
DGCNN [16]	1.0	36.9	2.4	168.1	48.0
RSNet [26]	6.9	2.2	1.1	111.5	52.0
SPVCNN (0.25×C)	0.2	0.4	1.1	21.1	50.4
PVCNN (0.25×C)	0.2	0.9	**0.7**	**9.0**	**52.3**
3D-UNet [2]	14.0	349.6	6.8	574.7	55.0
SPVCNN (1×C)	2.7	6.6	1.8	41.0	54.9
PVCNN (1×C)	2.6	13.0	1.3	**39.1**	56.1
PVCNN++ (0.5×C)	3.4	6.6	**0.7**	40.9	**57.6**
PointCNN [6]	11.5	17.5	4.6	282.3	57.3
PVCNN++ (1×C)	13.7	26.2	**0.8**	**67.2**	**59.0**

On average, PVCNN and PVCNN++ outperform point-based models with **8**× *speedup and* **3**× *memory reduction. They outperform the voxel-based baseline with* **14**× *speedup and* **10**× *memory reduction.*

TABLE 5
Results of Outdoor Scene Segmentation on SemanticKITTI (3D)

	#Params (M)	#MACs (G)	Latency (ms)	mIoU
PointNet [3]	3.0*	–	500*	14.6
SPGraph [23]	0.3*	–	5200*	17.4
PointNet++ [4]	6.0*	–	5900*	20.1
TangentConv [14]	0.4*	–	3000*	40.9
RandLA-Net [87]	1.2	66.5	880 (256+624)†	53.9
KPConv [88]	18.3	207.3	–	58.8
MinkowskiNet [32]	21.7	114.0	294	63.1
PVCNN	2.5	42.4	146	39.0
SPVCNN	21.8	118.6	317.1	65.3
SPVNAS-A	2.6	15.0	110	63.7
SPVNAS-B	12.5	73.8	259	**66.4**

SPVNAS outperforms MinkowskiNet with **2.7**× *speedup.* †*: computation time + post-processing time.* *: results from Behley* et al. *[75].*

averaged over at least three runs. For the other methods, we report the accuracy from their papers and measure #Params, #MACs and latency with their open-source implementation.

Results. As in Table 3, PVCNN achieves superior results compared with all manually-designed point-based methods. Specifically, PVCNN (0.5×C) outperforms PointCNN with **27.7**× model size reduction, **31.3**× computation reduction, and **23.7**× measured speedup. This indicates that PVConv scales much better (w.r.t. the number of points) than other point-based primitives. Then, we apply 3D-NAS to PVCNN under 4.2/4.9 ms latency constraints on NVIDIA GTX 1080 Ti GPU. The resulting PVNAS outperforms PVCNN (0.72×C) and PVCNN (1×C) with 1.3× and 1.4× speedup, respectively. It also compares favorably with AutoML-based approaches. With more than **29**× speedup, PVNAS achieves similar IoU compared with SGAS and LC-NAS.

PV/SPVConv. On PartNet, PVCNN (0.5×C) is **3.8**× faster than SPVCNN (1×C), while achieving **2.2%** higher accuracy. 3D-NAS improves the performance of SPVCNN significantly; however, SPVNAS is still **3.1**× slower than PVNAS-A. This again emphasizes the importance of primitive selection.

Deployment. We deploy our PVCNN and PVNAS on edge devices. As in Fig. 7a, PVCNN runs at real time (20 FPS) on NVIDIA Jetson Nano, which only consumes the power of a light bulb (5 W). Even when the number of input points increases by 5× on PartNet, PVNAS can still run at 8.3 FPS on NVIDIA Jetson Nano (Fig. 7b). Thus, PVNAS can empower efficient 3D vision on low-power devices, which becomes increasingly important with the rise of AR/VR applications.

6.2 3D Indoor Scene Segmentation

We then evaluate our method on 3D semantic segmentation and conduct experiments on the large-scale indoor scene dataset, S3DIS [79], [80]. Though indoor scenes are usually much larger (*e.g.*, 6m×6m×3m) than single 3D objects, it is still affordable to process them using sliding windows (*e.g.*, 1.5m×1.5m×3m). Note that the evaluation protocol is different from recent methods [32] that take in the entire scene. Our setting is closer to the real-world scenario since

depth cameras only capture a small region in a single pass. Similar to object part segmentation, we build PVCNN based on PointNet [3] and PVCNN++ based on PointNet++ [4].

Baselines. We compare our models with the state-of-the-art point-based models [3], [6], [16], [26] and the voxel-based baseline [2]. Following Tchapmi *et al.* [21] and Li *et al.* [6], we train the models on area 1,2,3,4,6 and test them on area 5 because it is the only one that does not overlap with any other area. Both data processing and evaluation protocol are the same as PointCNN [6] for fair comparison. We measure the latency and memory consumption with 32768 points per batch on a single NVIDIA GTX 1080 Ti GPU.

Results. As in Table 4, PVCNN improves its backbone (*i.e.*, PointNet) by more than **13%** in mIoU and outperforms DGCNN (which involves sophisticated graph convolutions) by a large margin in both accuracy and latency. Remarkably, our PVCNN++ outperforms the state-of-the-art point-based model (PointCNN) by 1.7% in mIoU with **4**× lower latency, and the voxel-based baseline (3D-UNet) by 4% in mIoU with more than **8**× lower latency and GPU memory consumption. Similar to object part segmentation, we also design compact models by reducing the number of channels in our PVCNN to 12.5%, 25% and 50% and our PVCNN++ to 50%. Remarkably, the narrower version of our PVCNN outperforms DGCNN with **15**× measured speedup, and RSNet with **9**× measured speedup. Furthermore, it achieves 4% improvement in mIoU over PointNet while still being **2.5**× faster than this extremely efficient 3D model (which does not have any neighborhood aggregation). We refer the readers to Fig. 8 for accuracy *vs.* latency and accuracy *vs.* memory trade-offs.

PV/SPVConv. With the same network structure, PVCNN is more effective than SPVCNN, achieving 1.2-2.1% higher mIoU (Table 4). Similar to our results on object segmentation, SPVCNN fails to achieve large speedups with small channel numbers: with 4× channel reduction, SPVCNN achieves only 1.9× speedup while PVCNN achieves 4.3× speedup. Thus, PVCNN is a superior choice for indoor scene segmentation.
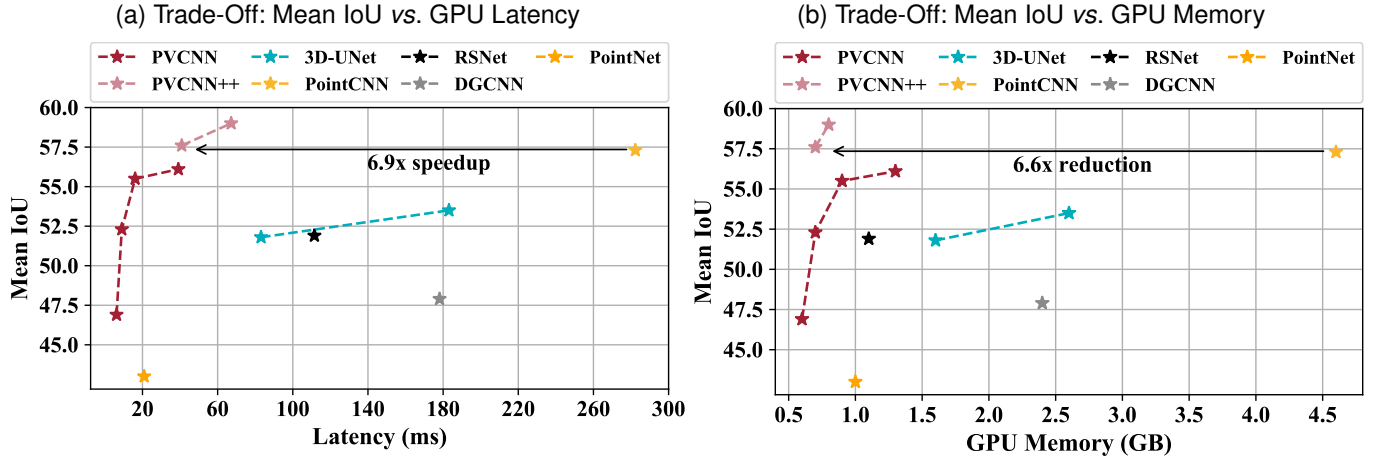
Fig. 8. PVCNN achieves a much better trade-off between accuracy and efficiency than the point-based and voxel-based baselines on S3DIS.
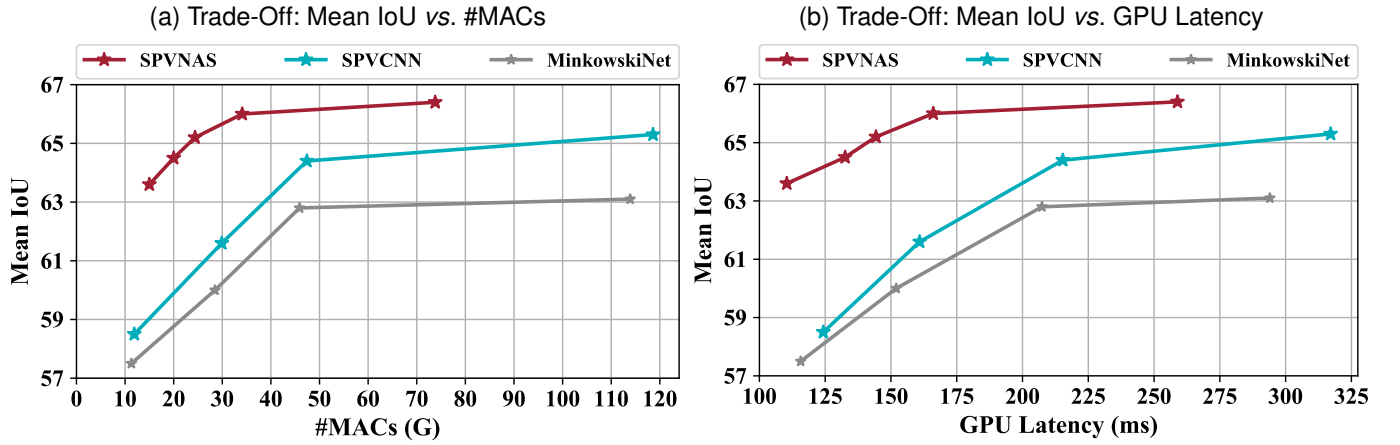

Fig. 9. An efficient 3D primitive (SPVConv) and a well-designed network architecture (3D-NAS) are both important to the performance of SPVNAS.
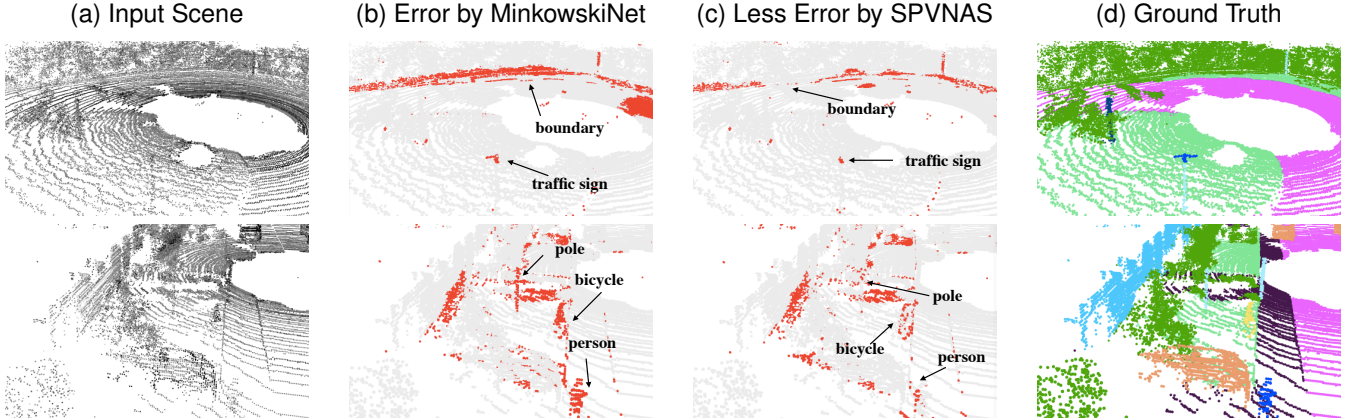

Fig. 10. MinkowskiNet usually has a higher error recognizing small objects and region boundaries, while our SPVNAS recognizes small objects better thanks to the high-resolution point-based branch.

6.3 3D Outdoor Scene Segmentation

6.3.1 SemanticKITTI

We evaluate our method on 3D semantic segmentation and conduct experiments on the large-scale outdoor scene dataset, SemanticKITTI [75]. This dataset contains 23,201 LiDAR point clouds for training and 20,351 for testing, and it is annotated

from all 22 sequences in the KITTI [93] Odometry benchmark. We train all models on the entire training set and report the mean intersection-over-union (mIoU) on the official test set under the single scan setting. We measure the latency on a single NVIDIA GTX 1080 Ti GPU with batch size of 1. We

TABLE 6
Results of Outdoor Scene Segmentation on SemanticKITTI (2D)

	#Params (M)	#MACs (G)	Latency (ms)	mIoU
DarkNet21Seg [75]	24.7	212.6	73 (49+24)†	47.4
DarkNet53Seg [75]	50.4	376.3	102 (78+24)†	49.9
SqueezeSegV3-21 [89]	9.4	187.5	97 (73+24)†	51.6
SqueezeSegV3-53 [89]	26.2	515.2	238 (214+24)†	55.9
3D-MiniNet [90]	4.0	–	–	55.8
PolarNet [91]	13.6	135.0	62	57.2
SalsaNext [92]	6.7	62.8	71 (47+24)†	59.5
SPVNAS	**1.1**	**8.9**	89	**60.3**

*SPVNAS outperforms the 2D projection-based methods with at least **7.1×** computation reduction. †: computation time + projection time.*

use `TorchSparse v1.0.0`* as our inference backend.

Results. As in Table 5, SPVNAS outperforms the previous state-of-the-art MinkowskiNet [32] by **3.3%** in mIoU with 1.7× model size reduction, 1.5× computation reduction and 1.1× measured speedup. Further, we downscale our SPVNAS by setting the resource constraint to 15G MACs. This offers us with a much smaller model that outperforms MinkowskiNet with **8.3×** model size reduction, **7.6×** computation reduction, and **2.7×** measured speedup. In Fig. 10, we also provide qualitative comparisons between SPVNAS and MinkowskiNet: our SPVNAS makes fewer errors for small instances.

In Table 6, we compare SPVNAS with 2D projection-based models. With a smaller backbone (by removing the decoder layers), SPVNAS outperforms DarkNets [75] by more than **10%** in mIoU with 1.2× speedup even though 2D CNNs are much better optimized by modern deep learning libraries. Compared with other 2D methods, our SPVNAS achieves at least **8.5×** model size reduction and **15.2×** computation reduction while being much more accurate. Furthermore, SPVNAS achieves higher mIoU than KPConv [88], which is the previous state-of-the-art point-based model, with **17×** model size reduction and **23×** computation reduction.

Analysis. In Fig. 9, we present both mIoU *vs.* #MACs and mIoU *vs.* latency trade-offs, where we uniformly scale the channel numbers in MinkowskiNet and SPVCNN down as our baselines. It can be observed that a better 3D module (SPVNAS) and a well-designed 3D network architecture (3D-NAS) are equally important to the final performance boost. Remarkably, SPVNAS outperforms MinkowskiNet by more than 6% in mIoU at 110 ms latency. Such a large improvement comes from non-uniform channel scaling and elastic network depth. In these manually-designed models (MinkowskiNet and SPVCNN), 77% of the total computation is distributed to the upsampling stage. With 3D-NAS, this ratio is reduced to 47-63%, making the computation more balanced and the downsampling stage (feature extraction) more emphasized.

6.3.2   nuScenes

The distribution of point clouds varies significantly across different sensors. To verify the generalizability of our SPVCNN and SPVNAS, we also conduct experiments on a recent 3D segmentation benchmark, nuScenes [81]. The dataset contains 34,149 LiDAR point clouds for training and 6,008 for testing,

*. https://github.com/mit-han-lab/torchsparse

TABLE 7
Results of Outdoor Scene Segmentation on nuScenes

	#Params (M)	#MACs (G)	Latency (ms)	mIoU
PolarSeg [91]	13.6	45.6	**42.8**	47.8
MinkowskiNet [32]	21.7	22.2	81.0	53.7
SPVCNN	21.8	23.1	89.9	54.7
SPVNAS	**9.6**	**10.6**	59.1	**54.7**

*SPVNAS outperforms MinkowskiNet with **2.1×** computation reduction and **1.4×** measured speedup.*

TABLE 8
Results of Outdoor Object Detection on KITTI (Two-Stage)

	Mem. (G)	Lat. (ms)	Car Easy	Mod.	Hard	Cyclist Easy	Mod.	Hard	Pedestrian Easy	Mod.	Hard
F-PN	1.3	29.1	85.2	71.6	63.8	77.1	56.5	52.8	66.4	56.9	50.4
F-PN++	2.0	105.2	84.7	72.0	64.2	75.6	56.7	53.3	68.4	60.0	52.6
PVCNN	1.4	58.9	**85.3**	**72.1**	**64.2**	**78.1**	**57.5**	**53.7**	**70.6**	**61.2**	**56.3**

*PVCNN outperforms F-PointNet++ in all categories significantly with **1.8×** measured speedup and **1.4×** memory reduction.*

TABLE 9
Results of Outdoor Object Detection on KITTI (One-Stage)

	Car Easy	Mod.	Hard	Cyclist Easy	Mod.	Hard	Pedestrian Easy	Mod.	Hard
SECOND [94]	89.8	80.9	78.4	82.5	62.8	58.9	**68.3**	60.8	55.3
SPVCNN	**90.9**	**81.8**	**79.2**	**85.1**	**63.8**	**60.1**	68.2	**61.6**	**55.9**

SPVCNN outperforms SECOND in most categories especially in cyclists.

and provides point-level semantic annotations for each point cloud. We follow the official instruction to split the training set into `train` split (consisting of 28,130 LiDAR point clouds) and `val` split (consisting of 6,019 LiDAR point clouds). We train all manually-designed models on the `train` split and evaluate them on the `val` split. For our SPVNAS, we search the best model on a 20% holdout `minival` split from the `train` split and perform evaluation on the `val` split. We report the official 31-class mIoU as the evaluation metric. Similar to SemanticKITTI, we measure the latency on an NVIDIA GTX 1080 Ti GPU with batch size of 1.

Results. As in Table 7, SPVNAS achieves a better accuracy *vs.* efficiency trade-off than both PolarSeg [91] and MinkowskiNet [32]. It outperforms MinkowskiNet by **1.0%** in mIoU with **2.3×** model size reduction, **2.1×** computation reduction, and **1.4×** speedup. It achieves **6.9%** mIoU improvement over the projection-based PolarSeg. Similar to our observations on SemanticKITTI, both efficient 3D primitive design (SPVConv, **+1.0** mIoU) and network architecture search (3D-NAS, **2.2×** computation reduction, **1.5×** measured speedup) contribute to the accuracy and efficiency boost of SPVNAS.

6.4   3D Outdoor Object Detection

We evaluate our method on 3D object detection and conduct experiments on the large-scale outdoor scene dataset, KITTI [93]. We follow the conventional training-validation
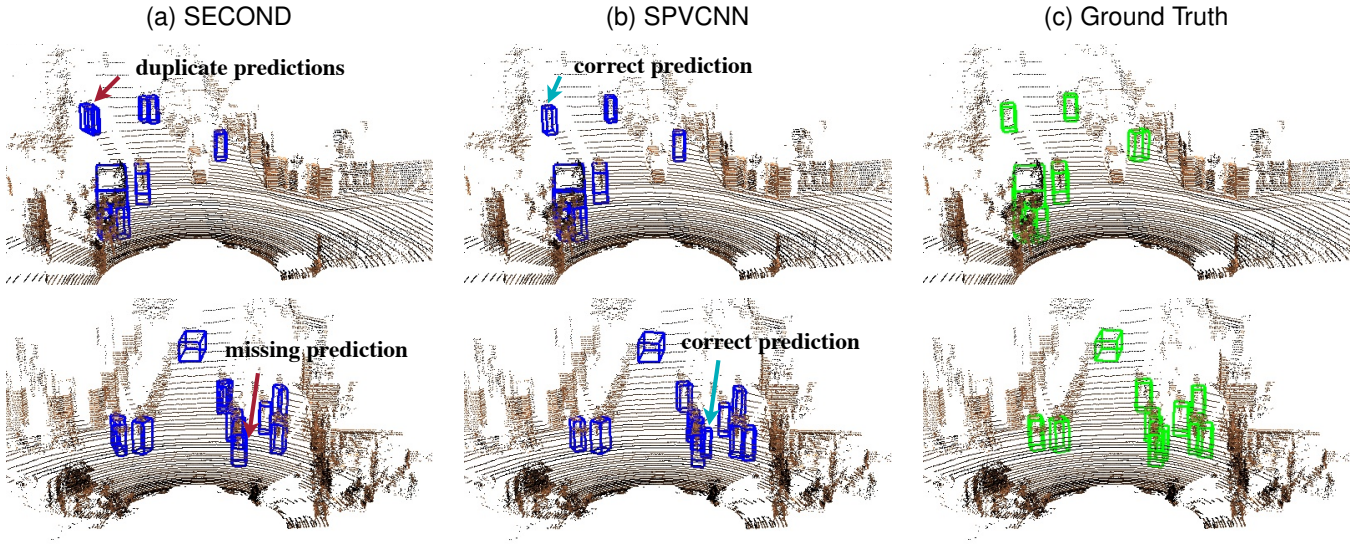
(a) SECOND (b) SPVCNN (c) Ground Truth

Fig. 11. Visualizations of outdoor object detection on KITTI. SPVCNN performs better than SECOND in detecting small objects in crowded scenes.
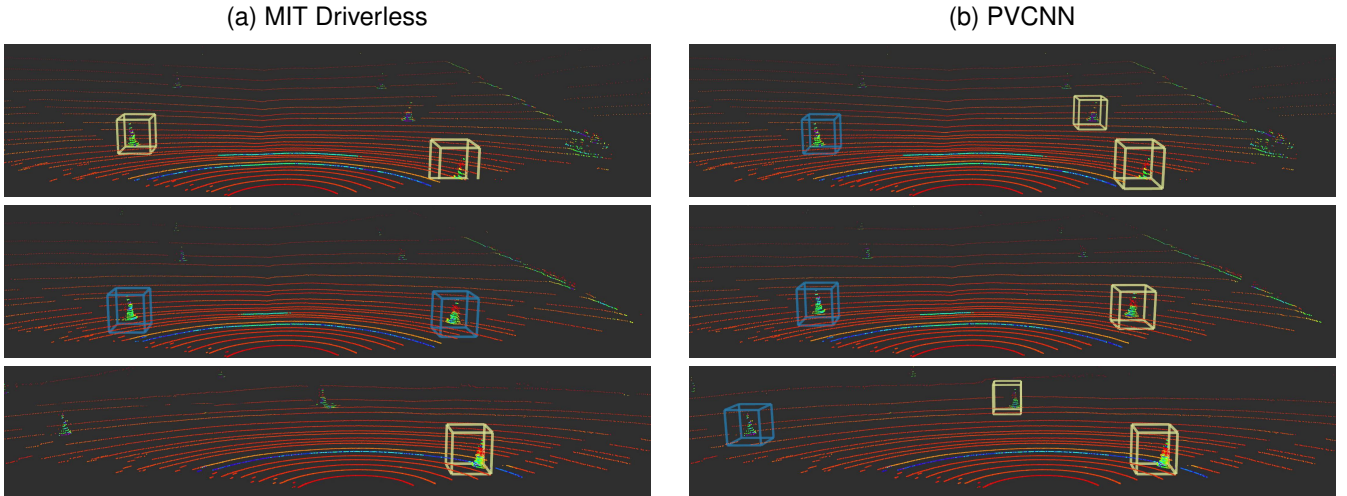
(a) MIT Driverless (b) PVCNN

Fig. 12. PVCNN is much more accurate: *e.g.*, MIT Driverless' model misclassifies the right traffic cone in the second example as blue. PVCNN also supports larger detection range (see the first and the third example), enabling the racing vehicle to confidently run at a higher speed.

split, where 3,712 samples are used for training and 3,769 samples are left for validation. We report the 3D average precision (with 40 recall positions) on the validation set under IoU thresholds of 0.7 for car, 0.5 for cyclist and pedestrian.

Results. We first apply our method to F-PointNet [27], which is a two-stage 3D object detection framework. It first leverages the off-the-shelf 2D object detector to produce frustrum proposals. For each frustrum, it then applies PointNets to classify the content. As the frustrums are relatively small, it is suitable to process them using PVConv. Thus, we build our PVCNN based on F-PointNet by replacing the MLP layers within its instance segmentation network with PVConv and keep its box proposal and refinement networks unchanged. We compare our PVCNN with F-PointNet (whose backbone is PointNet) and F-PointNet++ (whose backbone is Point-Net++). In Table 8, even if our PVCNN does not aggregate neighboring features in the box estimation network while F-PointNet++ does, PVCNN still outperforms it in all classes

with **1.8×** lower latency. Remarkably, our model achieves **2.4%** average mAP improvement in the most challenging pedestrian class. Compared with F-PointNet, our PVCNN obtains up to **4-5%** mAP improvement in pedestrians, which indicates that our model is both efficient and expressive.

Apart from the frustrum-based framework, we also apply our method to SECOND [94], which is a single-stage 3D object detection framework. It consists of a sparse encoder using 3D sparse convolutions and a region proposal network that performs 2D convolutions after projecting the encoded features into the bird's-eye view (BEV). Unlike F-PointNet, SECOND is applied to the entire outdoor scenes, which are of large scale. Therefore, we choose to replace the sparse encoder with SPVCNN. As a fair comparison, we reimplement and retrain SECOND, which already outperforms the results claimed in the original paper [94]. As summarized in Table 9, our SPVCNN achieves consistent improvements in all categories, especially in cyclist and pedestrian. This is
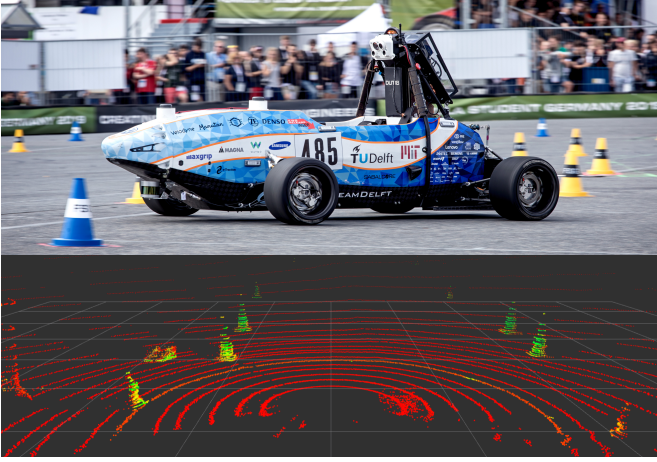
Fig. 13. Autonomous racing vehicle of MIT Driverless (at the Formula Student competition). Top: vehicle on the racing track (©FSG - Elena Schulz). Bottom: 3D data collected by the Velodyne 32-channel LiDAR sensor mounted on the car.

TABLE 10
Results of Cone Classification for Autonomous Racing Vehicle

	#Params (M)	#MACs (G)	Latency (ms)	Accuracy (%)
MIT Driverless	–	–	14.0	95.0
PointNet [3]	2.0	0.07	**1.4**	96.6
PointNet++ [4]	1.7	4.0	53.5	97.0
DGCNN [16]	0.6	0.03	2.1	98.0
PVCNN	**0.5**	**0.03**	2.3	**98.8**

Compared with existing learning-based solutions, PVCNN achieves higher accuracy with low latency. Here, we measure the GPU latency on NVIDIA GTX 1080 Ti based on an average of 8 cones per scene.

because the high-resolution point-based branch carries more information for small objects. We also provide qualitative results in Fig. 11, where SPVCNN excels in detecting small objects in crowded scenes.

7 APPLICATIONS

The autonomous racing vehicle is an ideal testbed of autonomous driving systems. Given the image and LiDAR inputs, the system needs to operate the car to drive on the racing track as fast as possible (and avoid the obstacles). As in Fig. 13, the racing track is surrounded by traffic cones in the Formula Student competition. In particular, there are three types of traffic cones, the blue ones representing the left boundary of the track, the yellow ones defining the right boundary of the track, and the orange ones marking the start and finish of the track. A typical solution is composed of perception, state estimation, path planning and control. Among these stages, the perception model serves as the eyes of the racing vehicle: it needs to localize and recognize the traffic cones accurately in order to segment the (drivable) racing track. The model needs to be very fast as its latency limits the velocity of the racing car; it also needs to be very accurate in order to prevent the car from driving outside the racing track and hitting the landmarks (2 seconds of penalty for each cone hit by the rules of the Formula Student competition). Therefore, we aim to improve the efficiency and accuracy of the LiDAR perception model with our PVCNN.

For the LiDAR perception pipeline, MIT Driverless adopts the clustering algorithm to localize the traffic cones on the racing track and then classifies the type of each traffic cone based on their LiDAR features. Due to the limited hardware resources on the car, their team previously designed a simple 1D feature-based neural network as their cone classification model. Though simple, it achieves moderate classification accuracy of 95%, which helped their team win the second and third places in two of the largest 2019 Formula Student competitions. However, this simple perception model can only support the detection range of 8 meters, limiting the velocity of their car to only 5 meters per second. Together with MIT Driverless, we replace their traffic cone classification model with our PVCNN. As in Table 10, our PVCNN accelerates their previous pipeline by **6×** while achieving almost perfect classification accuracy (**98.8%**). PVCNN also outperforms other learning-based solutions such as PointNet (+2.2%), PointNet++ (+1.8%) and DGCNN (+0.8%). In Fig. 12, we also visualize the traffic cone classification results of their original solution and our PVCNN, from which, our PVCNN is indeed more accurate and supports larger detection range, enabling the racing vehicle to run at a higher speed. MIT Driverless has deployed our PVCNN into their latest system.

8 CONCLUSION

In this paper, we studied 3D deep learning from the hardware efficiency perspective. We systematically analyzed the bottlenecks of previous point-based and voxel-based models and proposed a novel hardware-efficient 3D primitive to combine the best from both. We further enhanced this primitive with the sparse convolution to make it more effective for large (outdoor) scenes. Based on our designed 3D primitive, we then introduced the 3D neural architecture search framework to automatically explore the best network architecture that satisfies a given resource constraint. Extensive experiments on multiple 3D benchmark datasets validate the efficiency and effectiveness of our proposed method. Finally, we also deployed our model to the autonomous racing vehicle of MIT Driverless, achieving larger detection range, higher accuracy and lower latency. We hope that this paper will enable more real-world applications and inspire future research in the direction of efficient 3D deep learning.

REFERENCES
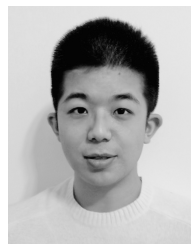
[1] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning Deep 3D Representations at High Resolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017.

[2] O. Cicek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation," in *Proc. Medical Image Computing and Computer Assisted Intervention*, 2016.

[3] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017.

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017.

[5] R. Klokov and V. S. Lempitsky, "Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017.

[6] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on $\mathcal{X}$-Transformed Points," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018.

[7] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A Deep Representation for Volumetric Shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015.

[8] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," *arXiv:1512.03012*, 2015.

[9] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2015.

[10] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and Multi-View CNNs for Object Classification on 3D Data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016.

[11] Z. Wang and F. Lu, "VoxSegNet: Volumetric CNNs for Semantic Part Segmentation of 3D Shapes," *IEEE Trans. Vis. Comput. Graph*, 2019.

[12] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[13] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis," *ACM Trans. Graph.*, 2017.

[14] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou, "Tangent Convolutions for Dense Prediction in 3D," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[15] T. Le and Y. Duan, "PointGrid: A Deep Network for 3D Shape Understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[16] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," *ACM Trans. Graph.*, 2019.

[17] S. Lan, R. Yu, G. Yu, and L. S. Davis, "Modeling Local Geometric Structure of 3D Point Clouds using Geo-CNN," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[18] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters," in *Proc. Eur. Conf. Comput. Vis.*, 2018.

[19] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, "SPLATNet: Sparse Lattice Networks for Point Cloud Processing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[20] J. Li, B. M. Chen, and G. H. Lee, "SO-Net: Self-Organizing Network for Point Cloud Analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[21] L. P. Tchapmi, C. B. Choy, I. Armeni, J. Gwak, and S. Savarese, "SEGCloud: Semantic Segmentation of 3D Point Clouds," in *Proc. Int. Conf. 3D Vis.*, 2017.

[22] W. Wang, R. Yu, Q. Huang, and U. Neumann, "SGPN: Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[23] L. Landrieu and M. Simonovsky, "Large-Scale Point Cloud Semantic Segmentation With Superpoint Graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[24] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun, "Deep Parametric Continuous Convolutional Neural Networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[25] B. Graham, M. Engelcke, and L. van der Maaten, "3D Semantic Segmentation With Submanifold Sparse Convolutional Networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[26] Q. Huang, W. Wang, and U. Neumann, "Recurrent Slice Networks for 3D Segmentation on Point Clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[27] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[28] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[29] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[30] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes," in *SIGGRAPH Asia*, 2018.

[31] H. Lei, N. Akhtar, and A. Mian, "Octree Guided CNN With Spherical Kernels for 3D Point Clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[32] C. Choy, J. Gwak, and S. Savarese, "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[33] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015.

[34] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *Proc. Int. Conf. Learn. Representations*, 2016.

[35] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices," in *Proc. Eur. Conf. Comput. Vis.*, 2018.

[36] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights," in *Proc. Int. Conf. Learn. Representations*, 2017.

[37] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-Aware Automated Quantization with Mixed Precision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[38] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and $<$ 0.5MB Model Size," *arXiv*, 2016.

[39] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv:1704.04861*, 2017.

[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[41] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019.

[42] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[43] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design," in *Proc. Eur. Conf. Comput. Vis.*, 2018.

[44] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *Proc. Int. Conf. Learn. Representations*, 2017.

[45] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[46] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive Neural Architecture Search," in *Proc. Eur. Conf. Comput. Vis.*, 2018.

[47] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[48] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-aware Efficient Convnet Design via Differentiable Neural Architecture Search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[49] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proc. Int. Conf. Mach. Learn.*, 2019.

[50] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "HAT: Hardware-Aware Transformers for Efficient Natural Language Processing," in *Proc. Annu. Meet. Assoc. Comput. Linguistics*, 2020.

[51] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," in *Proc. Annu. Meet. Assoc. Comput. Linguistics*, 2019.

[52] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *Proc. Int. Conf. Learn. Representations*, 2019.

[53] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware," in *Proc. Int. Conf. Learn. Representations*, 2019.

[54] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single Path One-Shot Neural Architecture Search with Uniform Sampling," in *Proc. Eur. Conf. Comput. Vis.*, 2020.

[55] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, "DetNAS: Backbone Search for Object Detection," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019.

[56] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for All: Train One Network and Specialize it for Efficient Deployment," in *Proc. Int. Conf. Learn. Representations*, 2020.

[57] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-Path NAS: Designing Hardware-Efficient ConvNets in Less Than 4 Hours," *arXiv:1904.02877*, 2019.

[58] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *Proc. Eur. Conf. Comput. Vis.*, 2018.

[59] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019.

[60] H. Cai, J. Lin, Y. Lin, Z. Liu, K. Wang, T. Wang, L. Zhu, and S. Han, "AutoML for Architecting Efficient and Specialized Neural Networks," *IEEE Micro*, 2019.

[61] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "GAN Compression: Efficient Architectures for Interactive Conditional GANs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[62] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Hardware-Centric AutoML for Mixed-Precision Quantization," *Int. J. Comput. Vis.*, 2020.

[63] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "APQ: Joint Search for Network Architecture, Pruning and Quantization Policy," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[64] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollar, "On Network Design Spaces for Visual Recognition," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019.

[65] Z. Zhu, C. Liu, D. Yang, A. Yuille, and D. Xu, "V-NAS: Neural Architecture Search for Volumetric Medical Image Segmentation," in *Proc. Int. Conf. 3D Vis.*, 2019.

[66] S. Kim, I. Kim, S. Lim, W. Baek, C. Kim, H. Cho, B. Yoon, and T. Kim, "Scalable Neural Architecture Search for 3D Medical Image Segmentation," in *Proc. Medical Image Computing and Computer Assisted Intervention*, 2019.

[67] D. Yang, H. Roth, Z. Xu, F. Milletari, L. Zhang, and D. Xu, "Searching Learning Strategy with Reinforcement Learning for 3D Medical Image Segmentation," in *Proc. Medical Image Computing and Computer Assisted Intervention*, 2019.

[68] W. Bae, S. Lee, Y. Lee, B. Park, M. Chung, and K.-H. Jung, "Resource Optimized Neural Architecture Search for 3D Medical Image Segmentation," in *Proc. Medical Image Computing and Computer Assisted Intervention*, 2019.

[69] K. C. Wong and M. Moradi, "SegNAS3D: Network Architecture Search with Derivative-Free Global Optimization for 3D Image Segmentation," in *Proc. Medical Image Computing and Computer Assisted Intervention*, 2019.

[70] Q. Yu, D. Yang, H. Roth, Y. Bai, Y. Zhang, A. Yuille, and D. Xu, "C2FNAS: Coarse-to-Fine Neural Architecture Search for 3D Medical Image Segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[71] Z. Ma, Z. Zhou, Y. Liu, Y. Lei, and H. Yan, "Auto-ORVNet: Orientation-Boosted Volumetric Neural Architecture Search for 3D Shape Classification," *IEEE Access*, 2020.

[72] G. Li, G. Qian, I. C. Delgadillo, M. Muller, A. Thabet, and B. Ghanem, "SGAS: Sequential Greedy Architecture Search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[73] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proc. Int. Conf. Mach. Learn.*, 2015.

[74] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *Proc. Int. Conf. Mach. Learn.*, 2013.

[75] 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. IEEE Int. Conf. Comput. Vis.*, 2019.

[76] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009.

[77] W. Wu, Z. Qi, and L. Fuxin, "PointConv: Deep Convolutional Networks on 3D Point Clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[78] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su, "PartNet: A Large-Scale Benchmark for Fine-Grained and Hierarchical Part-Level 3D Object Understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019.

[79] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3D Semantic Parsing of Large-Scale Indoor Spaces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016.

[80] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese, "Joint 2D-3D-Semantic Data for Indoor Scene Understanding," *arXiv:1702.01105*, 2017.

[81] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A Multimodal Dataset for Autonomous Driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[82] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012.

[83] E.-T. Le, I. Kokkinos, and N. J. Mitra, "Going Deeper with Lean Point Networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[84] G. Li, M. Müller, G. Qian, I. C. Delgadillo, A. Abualshour, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs Go as Deep as CNNs?" in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019.

[85] G. Li, G. Qian, I. C. Delgadillo, M. Müller, A. Thabet, and B. Ghanem, "SGAS: Sequential Greedy Architecture Search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[86] G. Li, M. Xu, S. Giancola, A. Thabet, and B. Ghanem, "LC-NAS: Latency Constrained Neural Architecture Search for Point Cloud Networks," *arXiv:2008.10309*, 2020.

[87] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[88] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "KPConv: Flexible and Deformable Convolution for Point Clouds," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019.

[89] C. Xu, B. Wu, Z. Wang, W. Zhan, P. Vajda, K. Keutzer, and M. Tomizuka, "SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation," in *Proc. Eur. Conf. Comput. Vis.*, 2020.

[90] I. Alonso, L. Riazuelo, L. Montesano, and A. C. Murillo, "3D-MiniNet: Learning a 2D Representation from Point Clouds for Fast and Efficient 3D LIDAR Semantic Segmentation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2020.

[91] Y. Zhang, Z. Zhou, P. David, X. Yue, Z. Xi, B. Gong, and H. Foroosh, "PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020.

[92] T. Cortinhal, G. Tzelepis, and E. E. Aksoy, "SalsaNext: Fast, Uncertainty-aware Semantic Segmentation of LiDAR Point Clouds for Autonomous Driving," *arXiv:2003.03653*, 2020.

[93] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *Int. J. Robot. Res.*, 2013.

[94] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors (Basel)*, 2018.

Zhijian Liu received the B.Eng. degree in computer science from Shanghai Jiao Tong University, China, in 2018, and the S.M. degree in electrical engineering and computer science from MIT, in 2020. He is working toward the Ph.D. degree at MIT, under the supervision of Prof. Song Han. His research focuses on efficient deep learning and its applications in computer vision and robotics.

Haotian Tang is a first-year Ph.D. student at MIT EECS, advised by Prof. Song Han. Previously, he received his B.Eng. degree from Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2020. His research interest is co-designing efficient machine learning algorithms and systems.

Shengyu Zhao is an undergraduate student at Institute for Interdisciplinary Information Sciences, Tsinghua University. He was a visiting researcher at MIT under the supervision of Prof. Song Han. His research focuses on efficient deep learning, autonomous driving, and generative models.

Kevin Shao is a second-year undergraduate student at MIT EECS. He is an undergraduate researcher working under Prof. Song Han, and his research interest is computer vision for autonomous vehicles.

Song Han is an assistant professor at MIT EECS Department. Dr. Han received the Ph.D. degree in Electrical Engineering from Stanford University and B.S. degree in Electrical Engineering from Tsinghua University. Dr. Han's research focuses on efficient deep learning computing at the intersection between machine learning and computer architecture. He proposed "Deep Compression" and the "Efficient Inference Engine" that impacted the industry. He is a recipient of NSF CAREER Award, MIT Technology Review Innovators Under 35, best paper awards at ICLR 2016 and FPGA 2017, Facebook Faculty Award, SONY Faculty Award, AWS Machine Learning Research Award.