FlatFormer: Flattened Window Attention for Efficient Point Cloud Transformer

Zhijian Liu[1,*] Xinyu Yang[1,2,*] Haotian Tang[1] Shang Yang[1,3] Song Han[1]
[1]MIT [2]Shanghai Jiao Tong University [3]Tsinghua University

https://flatformer.mit.edu

2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) | 979-8-3503-0129-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/CVPR52729.2023.00122

Abstract

*Transformer, as an alternative to CNN, has been proven effective in many modalities (*e.g.*, texts and images). For 3D point cloud transformers, existing efforts focus primarily on pushing their accuracy to the state-of-the-art level. However, their latency lags behind sparse convolution-based models (**3× slower**), hindering their usage in resource-constrained, latency-sensitive applications (such as autonomous driving). This inefficiency comes from point clouds' sparse and irregular nature, whereas transformers are designed for dense, regular workloads. This paper presents **FlatFormer** to close this latency gap by trading spatial proximity for better computational regularity. We first flatten the point cloud with window-based sorting and partition points into **groups of equal sizes** rather than **windows of equal shapes**. This effectively avoids expensive structuring and padding overheads. We then apply self-attention within groups to extract local features, alternate sorting axis to gather features from different directions, and shift windows to exchange features across groups. FlatFormer delivers state-of-the-art accuracy on Waymo Open Dataset with **4.6**× speedup over (transformer-based) SST and **1.4**× speedup over (sparse convolutional) CenterPoint. This is the first point cloud transformer that achieves real-time performance on edge GPUs and is faster than sparse convolutional methods while achieving on-par or even superior accuracy on large-scale benchmarks.*

1. Introduction

Transformer [75] has become the model of choice in natural language processing (NLP), serving as the backbone of many successful large language models (LLMs) [2, 17]. Recently, its impact has further been expanded to the vision community, where vision transformers (ViTs) [18, 45, 74] have demonstrated on-par or even superior performance compared with CNNs in many visual modalities (*e.g.*, image and video). 3D point cloud, however, is not yet one of them.

∗ indicates equal contributions.

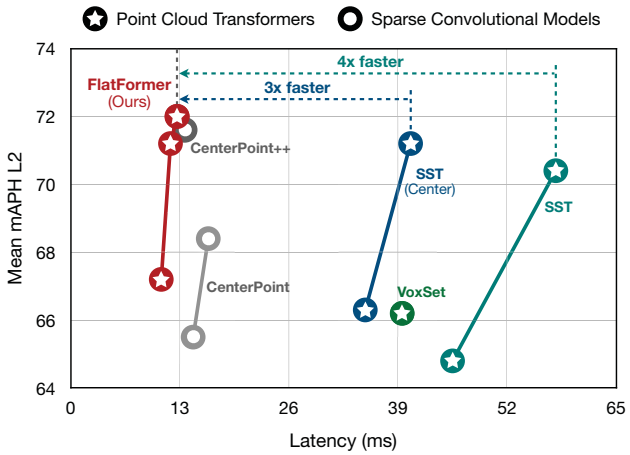

Figure 1. Previous point cloud transformers (⋆) are **3-4**× slower than sparse convolution-based models (●) despite achieving similar detection accuracy. FlatFormer is the first point cloud transformer that is faster than sparse convolutional methods with on-par accuracy. Latency is measured on an NVIDIA Quadro RTX A6000.

Different from images and videos, 3D point clouds are intrinsically sparse and irregular. Most existing point cloud models [94] are based on 3D sparse convolution [24], which computes convolution only on non-zero features. They require dedicated system support [14, 71, 91] to realize high utilization on parallel hardware (*e.g.*, GPUs).

Many efforts have been made toward point cloud transformers (PCTs) to explore their potential as an alternative to sparse convolution. Global PCTs [26] benefit from the regular computation pattern of self-attention but suffer greatly from the quadratic computational cost (w.r.t. the number of points). Local PCTs [50, 98] apply self-attention to a local neighborhood defined in a similar way to point-based models [57] and are thus bottlenecked by the expensive neighbor gathering [47]. These methods are only applicable to single objects or partial indoor scans (with <4k points) and cannot be efficiently scaled to outdoor scenes (with >30k points).

Inspired by Swin Transformer [45], window PCTs [19, 70] compute self-attention at the window level, achieving im-

979-8-3503-0129-8/23/$31.00 ©2023 IEEE
DOI 10.1109/CVPR52729.2023.00122

pressive accuracy on large-scale 3D detection benchmarks. Despite being spatially regular, these windows could have drastically different numbers of points (which differ by more than **80**×) due to the sparsity. This severe imbalance results in redundant computation with inefficient padding and partitioning overheads. As a result, window PCTs can be **3**× **slower** than sparse convolutional models (Figure 1), limiting their applications in resource-constrained, latency-sensitive scenarios (*e.g.*, autonomous driving, augmented reality).

This paper introduces **FlatFormer** to close this huge latency gap. Building upon window PCTs, FlatFormer trades spatial proximity for better computational regularity by partitioning 3D point cloud into *groups of equal sizes* instead of *windows of equal shapes*. It applies self-attention within groups to extract local features, alternates the sorting axis to aggregate features from different orientations, and shifts windows to exchange features across groups. Benefit from the regular computation pattern, FlatFormer achieves **4.6**× speedup over (transformer-based) SST and **1.4**× speedup over (sparse convolutional) CenterPoint while delivering the state-of-the-art accuracy on Waymo Open Dataset.

To the best of our knowledge, FlatFormer is the first point cloud transformer that achieves on-par or superior accuracy than sparse convolutional methods with lower latency. It is also the first to achieve real-time performance on edge GPUs. With better hardware support for transformers (*e.g.*, NVIDIA Hopper), point cloud transformers will have a huge potential to be the model of choice in 3D deep learning. We believe our work will inspire future research in this direction.

2. Related Work

Deep Learning on Point Clouds. Early research converts point clouds from 3D sensors to dense voxel grids and applies 3D CNNs [15,51,56,87] on the volumetric inputs. However, the compute and memory consumption of volumetric CNNs grows cubically w.r.t. the input resolution, limiting the scalability of these methods. To overcome this bottleneck, later research [28, 35, 55, 57, 59, 73, 83, 85] directly performs feature extraction on point sets, while [60, 76, 77] convert point clouds to octrees and [11, 14, 24, 43, 91] perform sparse convolution on sparse voxels. Recently, researchers also explore point+voxel [47–49, 89] or point+sparse voxel [49, 63, 64, 72] hybrid representations for efficient 3D deep learning.

3D Object Detection. Extensive attention has been paid to 3D object detection [3, 23, 69] for autonomous vehicles. Early research [54, 84] generates object proposals on 2D images and refines the predictions in the lifted 3D frustums. VoxelNet [101] leads another line of research that directly detects 3D objects without 2D proposals. Following VoxelNet, PointPillars [34], SECOND [91, 103], 3DSSD [92] and MVF [100] are all single-stage anchor-based 3D detectors, while CenterPoint [94, 96], AFDet [22, 29], HotspotNet [5], MVF++ [58], RangeDet [21], PolarStream [6], ObjectDGCNN [82], M3DETR [25], PillarNet [62], LidarMultiNet [93] are single-stage anchor-free 3D detectors. PointRCNN [65], Fast Point R-CNN [12], Part-A^2Net [66], PVRCNN [63, 64], LiDAR R-CNN [39], CenterFormer [102], FSD [20], MPPNet [8] add a second stage that refines the proposals from the region proposal network (RPN) in the 3D space. There are also recent explorations on multi-sensor 3D object detection [1, 7, 10, 38, 42, 46, 61, 95].

Vision Transformers. Motivated by the huge success of transformers [17, 75] in natural language processing (NLP), researchers have started to adapt transformers to various vision tasks [32]. The pioneering ViT [33] first demonstrates that an image can be viewed as 16×16 words and processed by multi-head attention layers. DeiT [74] further shows that ViTs can be trained in a data-efficient manner without pretraining on JFT [68]. T2T-ViT [97], Pyramid ViT [78, 79] and CrossFormer [80] introduce hierarchical modeling capability to ViTs. Swin Transformer [44,45] limits self-attention computation to non-overlapping windows and enables cross-window information exchange via window shifting. There are also task-specific ViTs such as ViTDet [37] for object detection, SETR [99], and SegFormer [88] for semantic segmentation. Instead of adopting a fully-transformer backbone, another line of research, such as DETR [4], Deformable DETR [104], MaskFormer [13], PanopticSegFormer [41], DETR3D [81], BEVFormer [40], apply self-attention only to the task-specific heads and still uses CNNs for the backbone.

Point Cloud Transformers. Recently, fully-transformer architectures have begun to emerge in the point cloud domain. Similar to ViT, PCT [26] calculates self-attention globally on the entire point cloud, which falls short in scalability as its computation complexity scales quadratically as the number of points grows. PointASNL [90], PointTransformer [86, 98], Fast Point Transformer [53], PointFormer [52], VoTr [50], VoxSet [27] applies transformer-based architecture on the local neighborhood of each point. The efficiency of these local transformers is limited by neighborhood query and feature restructuring. Most related to our work are the window-based point cloud transformers, SST [19] and SWFormer [70]. Inspired by Swin Transformer, they project the point cloud into a bird's-eye view and divide the BEV space into non-overlapping windows with the same *spatial sizes* (but different *number of points*). Window shifting is used to communicate information across windows. SST suffers from large computation in window partition and padding overhead due to regional grouping, and achieves only **one-sixth** utilization compared with sparse convolutional models.

In this paper, we only refer to those methods that adopt a transformer-based architecture in the *backbone* as point cloud transformers. As CenterFormer [102], FUTR3D [9] and UVTR [36] apply sparse convolutional backbones and

only use the transformer in their detection heads, we still categorize them as sparse convolutional methods.

3. Why are Point Cloud Transformers Slow?

Although point cloud transformers (PCTs) start to catch up with the accuracy of sparse convolutional detectors, there is still a 3× latency gap between the fastest PCT (SST [19]) and sparse convolutional CenterPoint [94] (Figure 1). In this section, we dissect the efficiency bottleneck of PCTs, which lays a solid foundation for our FlatFormer design.

3.1. Global PCTs

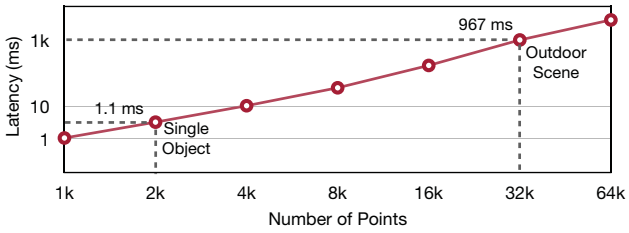

Figure 2. Latency of global PCTs scale quadratically with respect to the number of input points and cannot scale up to outdoor scenes.

Inspired by ViT [33], the most simple and straightforward design for transformers on point cloud is global PCTs [26], where each point is a token. They leverage multi-head self-attention (MHSA) [75] globally across the entire point cloud. While being effective on small-scale 3D objects, global PCTs fall short in scaling to large-scale scenes due to its $\mathcal{O}(N^2D)$ complexity, where N is the number of tokens and D is the number of channels. From Figure 2, the runtime of global PCTs [26] grows quadratically as the number of input points grows. For example, with 32k input points*, the model takes **almost one second** to execute on an NVIDIA A6000 GPU, **66**× slower than CenterPoint [94].

3.2. Local PCTs

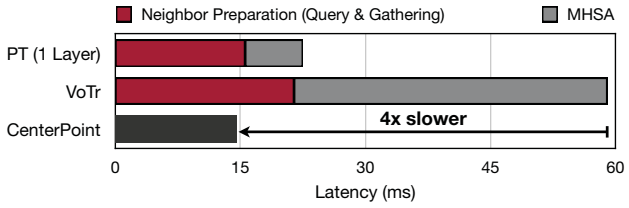

Figure 3. Local PCTs suffer from large neighborhood query and data restructuring overhead.

Researchers have proposed local PCTs [50, 52, 53, 86, 90, 98] to solve the scalability issue of global PCTs. They apply MHSAs to the neighborhood of each point rather than the entire point cloud. Hence, their computational complexity is $\mathcal{O}(NK^2D)$, where N is the number of points, K is the number of neighbors for each point, and D is the number of channels. As N ranges from 20k to 100k for real workloads and K is less than 64 for local PCTs, their theoretical cost is much lower than global PCTs.

Local PCTs, however, suffer greatly from neighbor preparation overheads. As point cloud is sparse and irregular, we have to first *find the neighbors* of each point, and then re-*structure the data* from the $N \times D$ format to the $N \times K \times D$ format on which MHSAs can be applied. These two steps are slow, taking **22** ms (*i.e.*, **36%** of the total runtime) for VoTr [50] to execute for a single scene on Waymo, which is already slower than the entire CenterPoint model. For Point Transformer (PT) [98], the cost for preparing neighbors takes up to **70%** of the runtime. Such overhead in a *single* layer is already larger than the total runtime of CenterPoint!

3.3. Window PCTs

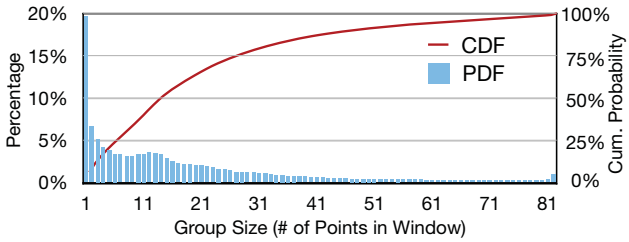

Figure 4. In SST [19], the number of points within each window has a large variance. Therefore, padding is necessary and leads to significant overhead for MHSA computation.

The great success of Swin Transformers [44, 45] in various visual recognition tasks motivates the design of window PCTs, among which, SST [19] is a representative work. It first projects the point cloud into the bird's-eye-view (BEV) space, then divides the BEV space into equally-shaped, non-overlapping windows, and applies MHSA within each window. Similar to Swin Transformer, SST uses window shifting to enable information exchange across windows.

Different from images, point clouds are sparse and non-uniformly distributed over the space. As a result, the number of point within each window is not the same and can differ by two orders of magnitude (Figure 4). As the vanilla MHSA kernel cannot efficiently support variable sequence lengths, SST [19] batches windows with similar sizes together and pad all windows in each batch to the largest group size within the batch (Figure 5l). It then applies MHSA within each batch separately. In practice, such padding introduces a **1.7**× computation overhead on Waymo. Worse still, partitioning points to equal windows also introduce significant latency overhead: it takes **18 ms** per scene on Waymo, even slower than the total runtime of CenterPoint. To sum up, the padding and partitioning overheads make SST less hardware-friendly

*32k is the number of points left after 0.32m×0.32m BEV projection in a single-frame Waymo [69] scene.

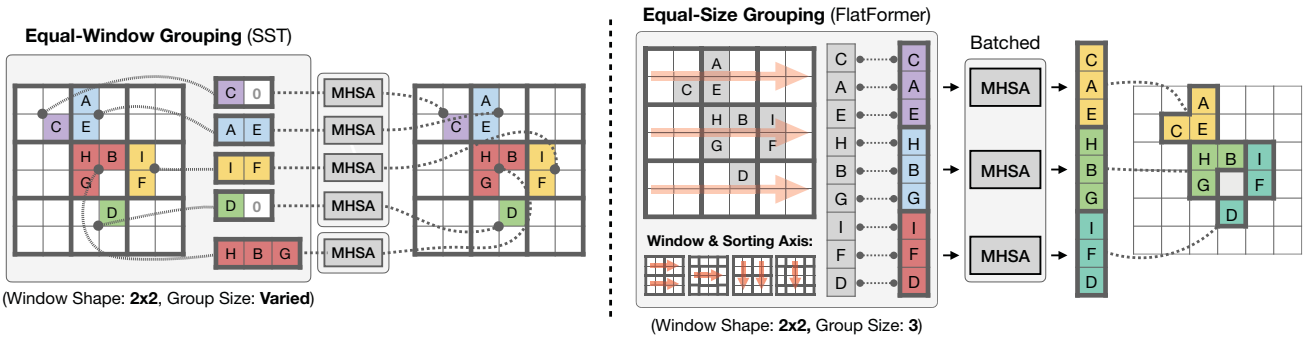

Figure 5. FlatFormer partitions the point cloud into groups of equal sizes (*right*), rather than windows of equal shapes (*left*). This effectively trades *spatial proximity* for better *computational regularity*. It then applies self-attention within each group to extract local features, alternates the sorting axis to aggregate features from different directions, and shifts windows to exchange features across groups.

compared with sparse convolutional methods.

4. FlatFormer

With all the lessons learned in Section 3, we will design our point cloud transformer to be scalable and efficient.

4.1. Overview

The basic building block of FlatFormer is Flattened Window Attention (FWA). As in Figure 5r, FWA adopts *window-based sorting* to flatten the point cloud and partitions it to *groups of equal sizes* rather than *windows of equal shapes*. This naturally resolves the group size imbalance problem and avoids the padding and partitioning overheads. FWA then applies self-attention within groups to extract local features, alternates sorting axis to aggregate features from different orientations, and shifts windows to exchange features across groups. Finally, we provide an implementation of FWA that further improves its efficiency and minimizes the overheads.

4.2. Flattened Window Attention (FWA)

4.2.1 Sorting & Grouping

Window-Based Sorting. With a point cloud $\{(x, y)\}$[†], we first quantize the coordinate of each point (x, y) to

$$\big(\underbrace{\lfloor x/w_x \rfloor, \lfloor y/w_y \rfloor}_{\text{window coordinates}}, \underbrace{x - \lfloor x/w_x \rfloor \cdot w_x, y - \lfloor y/w_y \rfloor \cdot w_y}_{\text{local coordinates within window}}\big), \tag{1}$$

where (w_x, w_y) is the window shape. Next, we sort all points first by *window coordinates* and then by *local coordinates* within the window. This step turns the unordered point cloud into an ordered one, where points within the same window will be next to each other.

Equal-Size Grouping. Conventional window PCTs [19] will then group the points within the same window together. However, as discussed in Section 3, each group can have drastically different numbers of points due to inherent sparsity. To overcome the padding overheads, we partition the point cloud into *groups of equal sizes* based on the sorted sequence. This step allows the subsequent group attention to enjoy a perfectly regular workload. From the implementation perspective, our grouping only involves a simple tensor reshaping (which is free since it does not change the layout) and is more efficient than window partitioning in SST [19].

Alternate Sorting Axis. Between the two axes, x has a higher priority in sorting. Thus, points with identical $\lfloor x/w_x \rfloor$ will be next to each other while points with the same $\lfloor y/w_y \rfloor$ can be very far away from each other in the sorted sequence, breaking the geometric locality. To solve this inequity, we alternate the sorting axis between x and y in different FWA blocks. This is very similar to spatially separable convolution that decomposes a 3×3 kernel into 3×1 and 1×3 kernels. Stacking FWA blocks with different sorting axes enables the model to aggregate local features from different directions.

Equal Size *vs*. Equal Window. The key design choice we made is to partition the point cloud into groups of equal sizes rather than windows of equal shapes. There is a trade-off: equal-window grouping maintains perfect *spatial proximity* (*i.e*., each group has the same radius) but breaks the *computational regularity*, while equal-size grouping ensures balanced computation workload (*i.e*., each group has the same number of points) but cannot guarantee the geometric locality. We show in Section 5.3 that computation regularity is more important since spatial irregularity can be partially addressed by our algorithm design: *i.e*., window-based sorting offers a fairly good spatial ordering, and self-attention is robust to outliers (*i.e*., distant point pairs).

4.2.2 Group Attention

With points partitioned, we then apply self-attention [75] within each group to extract local features. For each group

[†]We assume that the point cloud is in 2D for ease of notation, while our method applies to 3D or higher-dimension point clouds.

of points with coordinates $\mathcal{C}$ and features $\mathcal{F}$, we follow the standard transformer block design:

$$\begin{aligned} \mathcal{F}' &= \mathcal{F} + \texttt{MHSA}(\texttt{LN}(\mathcal{F}), \texttt{PE}(\mathcal{C})), \\ \mathcal{F}'' &= \mathcal{F}' + \texttt{FFN}(\texttt{LN}(\mathcal{F}')), \end{aligned} \tag{2}$$

where $\texttt{MHSA}(\cdot)$, $\texttt{FFN}(\cdot)$ and $\texttt{LN}(\cdot)$ correspond to multi-head self-attention, feed-forward layer, and layer normalization, respectively. Different from SST [19], $\texttt{PE}(\cdot)$ gives global absolute positional embedding. Here, we use the most standard `softmax` attention formulation for $\texttt{MHSA}(\cdot)$. Our method will benefit from other more efficient attention variants, such as linear attention [30], which we leave for future work.

Window Shift. Benefit from the non-overlapping design, window-based attention typically has a larger receptive field than convolution (*e.g.*, 69 neighbors in our FWA *vs.* $\leq$27 neighbors in a sparse convolution of kernel size 3). However, its modeling power is limited as there is no feature exchange across groups. Similar to Swin Transformer [19, 44, 45], we adopt the shifted window approach that alternates the sorting configuration in consecutive FWA blocks. Specifically, we translate the coordinates of all points by $(w_x/2, w_y/2)$ for sorting in shifted FWA blocks. This mechanism introduces cross-group feature communication while effectively maintaining workload independence between groups. Note that alternating sorting axis also enables feature exchange.

4.3. Efficient Implementation

Besides the algorithm design, we also provide an implementation that improves the efficiency of MHSA and FFN and minimizes the sorting and masking overheads. All these optimizations are specialized for our point cloud transformer design and are not applicable to sparse convolution models.

Efficient MHSA. Within MHSA, query $\mathcal{Q}$, key $\mathcal{K}$, and value $\mathcal{V}$ will first be transformed with separate linear layers. We pack these three linear projections into a batched matrix multiplication (since $\mathcal{Q}$, $\mathcal{K}$ and $\mathcal{V}$ have the same shape in our FWA) to improve the parallelism. In addition, standard attention implementations materialize $\mathcal{Q}\mathcal{K}^{\mathrm{T}}$ and $\texttt{softmax}(\mathcal{Q}\mathcal{K}^{\mathrm{T}})$. We leverage a recent efficient functional-preserving implementation (FlashAttention [16]) that uses tiling to reduce the number of memory reads/writes, achieving better efficiency.

Efficient FFN. FFN consists of two linear layers with a `GELU` activation in the middle. We implement a fused linear kernel (in `Triton`) that absorbs the activation into the layer before to avoid writing the intermediate results to DRAM. We also observe that our linear kernel (optimized by `Triton`) is even more efficient than `cuBLAS`, which is probably due to the unconventional tall-and-skinny workload.

Reuse Sorting. Sorting the coordinates of all points is a non-negligible overhead. As the coordinates remain identical (w/o downsampling), we reuse the sorting results (*i.e.*, ranks of each point) with the same axis and window. In practice, this reduces the sorting overhead in our model by 50%.

Drop Residual. The size of the input point cloud might not be divisible by the group size, generating a group with fewer points after partition. This minor irregularity will still result in some overheads in self-attention since we need to introduce masking to correctly zero them out. Instead, we directly drop the final non-full group. This only corresponds to less than 0.1% of all points, having a negligible impact on the model's performance (<0.1%).

5. Experiments

5.1. Setup

Dataset. We carry out our experiments on the large-scale Waymo Open Dataset (WOD) [69] with 1150 LiDAR point cloud sequences. Each sequence has 200 frames, collected by a 360° FoV LiDAR sensor at 10 frames per second. There are four foreground classes, three of which (vehicles, pedestrians and cyclists) are used for detection metric evaluation.

Metrics. We follow the official metrics on Waymo to calculate the standard 3D mAP and heading-weighted 3D mAP (mAPH) of all methods. The matching IoU thresholds for vehicle, pedestrian and cyclist are set to default values (0.7, 0.5 and 0.5). Objects are divided into two difficulty levels, where objects with fewer than five laser points or annotated as hard are categorized into level 2 (L2) and other objects are defined as level 1 (L1). We mainly report L2 metrics in the main paper and provide detailed metrics in the appendix.

Model. Based on FWA, we provide an instantiation of FlatFormer for 3D object detection. We follow the design of PointPillars [34] to first voxelize the point cloud into sparse BEV pillars (with MLPs) at a resolution of 0.32m×0.32m. We then apply eight consecutive FWA blocks with alternating sorting axes (*i.e.*, x or y) and window shifting configurations (*i.e.*, on or off). All FWA blocks have a window shape of 9×9 and a group size of 69. Following SST [19], we do not apply any spatial downsampling, which is beneficial for small objects. Finally, we apply regular BEV encoder and a center-based detection head following CenterPoint [94, 96].

5.2. Main Results

5.2.1 Single-Stage Detectors

Baseline. We compare our FlatFormer with state-of-the-art sparse convolutional [62, 94, 96] and transformer-based [19, 27, 50] single-stage 3D detectors. All models apply anchor- or center-based detection heads [91, 94, 96]. We compare models with different numbers of input frames separately.

Latency. We measure the latency on an NVIDIA Quadro RTX A6000 GPU using FP16 precision. We adopt SpConv v2.2.3 [91], the state-of-the-art 3D sparse convolution library,

	#Frames	#MACs (G)	Latency (ms)	Speedup (w.r.t. [94])	Mean L2 (mAPH)	Vehicle L2 (mAP/APH)	Pedestrian L2 (mAP/APH)	Cyclist L2 (mAP/APH)
SECOND [91][3]	1	–	–	–	57.2	63.9 / 63.3	60.7 / 51.3	58.3 / 57.0
PointPillars [34][3]	1	–	–	–	57.8	63.6 / 63.1	62.8 / 50.3	61.9 / 59.9
○ CenterPoint [94][1]	1	126.9	14.6	1.0×	65.5	66.7 / 66.2	68.3 / 62.6	68.7 / 67.6
● VoTr-SSD [50]	1	110.3	59.1*	0.2×	–	60.2 / 59.7	–	–
● SST [19][2]	1	204.9	45.5	0.3×	64.8	64.8 / 64.4	71.7 / 63.0	68.0 / 66.9
● SST-Center [19]	1	226.4	35.1	0.4×	66.3	66.6 / 66.2	72.4 / 65.0	68.9 / 67.6
● VoxSet [27]	1	189.4	39.5	0.4×	66.2	66.0 / 65.6	72.5 / 65.4	69.0 / 67.7
○ PillarNet [62]	1	138.3	11.1	1.3×	**67.2**	70.4 / 69.9	71.6 / 64.9	67.8 / 66.7
● **FlatFormer** (Ours)	1	177.2	**10.8**	**1.4**×	**67.2**	69.0 / 68.6	71.5 / 65.3	68.6 / 67.5
○ CenterPoint [94][1]	2	137.6	16.4	1.0×	68.4	67.7 / 67.2	71.0 / 67.5	71.5 / 70.5
○ PillarNet [62]	2	148.8	**11.6**	**1.4**×	70.0	71.6 / 71.1	74.5 / 71.4	68.3 / 67.5
● **FlatFormer** (Ours)	2	186.6	11.9	**1.4**×	**71.2**	70.8 / 70.3	73.8 / 70.5	73.6 / 72.6
○ CenterPoint [94]	3	144.7	18.3	1.0×	–	–	–	–
○ CenterPoint++ [96][1]	3	113.0	13.6	1.3×	71.6	71.8 / 71.4	73.5 / 70.8	73.7 / 72.8
● SST [19][2]	3	250.0	57.8	0.3×	70.4	66.5 / 66.1	76.2 / 72.3	73.6 / 72.8
● SST-Center [19]†	3	243.1	40.5	0.5×	71.2	68.8 / 68.2	75.8 / 71.8	74.4 / 73.3
● **FlatFormer** (Ours)	3	193.2	**12.7**	**1.4**×	**72.0**	71.4 / 71.0	74.5 / 71.3	74.7 / 73.7

Table 1. Results of single-stage 3D detectors on Waymo Open Dataset (validation set). FlatFormer achieves 1.4× speedup over CenterPoint and 4.6× speedup over SST while being more accurate. We refer the readers to the appendix for detailed metrics (*e.g.*, L1 mAP/mAPH). Markers ○ and ● refer to sparse convolutional models and point cloud transformers, respectively. Methods with <60 L2 mAPH are marked gray. (†: reproduced by us, [1]: from CenterPoint authors, [2]: from SST authors, [3]: from FSD paper, *: projected latency)

	#Frames	Latency (ms)	Mean L2 (mAPH)
○ LiDAR R-CNN [39]†	1	–	61.3
○ PV-RCNN [63]†	1	–	63.3
○ Part-A^2 [67]†	1	–	63.8
○ PV-RCNN++ [64]†	1	–	64.9
○ CenterFormer [102]	1	33.8	69.0
○ FSD-SpConv [20]	1	47.8	**70.8**
● **FlatFormer**+FSD (Ours)	1	39.3	70.5
○ CenterFormer [102]	2	53.5	72.8
● **FlatFormer**+FSD (Ours)	2	51.8	**73.8**
○ CenterFormer [102]	4	85.8	73.2
○ MPPNet [8]	4	–	74.2
● **FlatFormer**+FSD (Ours)	3	60.6	**74.8**

Table 2. Results of two-stage 3D detectors on Waymo Open Dataset (validation set). FlatFormer achieves on-par or even higher accuracy compared with sparse convolutional two-stage detectors. We refer the readers to the appendix for detailed metrics (*e.g.*, per-class L1/L2 mAP/mAPH). Markers ○ and ● refer to SpConv-based models and point cloud transformers, respectively. (†: from FSD paper)

to execute the 3D encoder of all sparse convolutional detectors. For transformer-based detectors, we use their official implementation to measure the runtime. All modules after the 3D encoder (*e.g.*, BEV encoder and detection head) are executed with TensorRT 8.4. We execute all the methods on the first 1,000 samples for 50 runs (with 10 warmup runs). We report the average latency (with outliers excluded). We do not include the data loading and post-processing time.

Results. As in Table 1, our FlatFormer achieves consistent performance improvements over both sparse convolutional and transformer-based detectors with much better efficiency. For one-frame models, FlatFormer is **4.2**×, **3.3**× and **3.7**× faster than SST, SST-Center and the recent VoxSet [27]. It also compares favorably with strong sparse convolutional baselines: **1.4**× faster than CenterPoint with 1.7 higher L2 mAPH and performs on par with PillarNet [62]. The accuracy advantage magnifies in the two-frame setting. Specifically, FlatFormer is 1.4× faster than CenterPoint with 2.8 L2 mAPH higher accuracy, and outperforms PillarNet by 1.3 L2 mAPH with a similar latency. With three input frames, FlatFormer is **4.6**× and **3.2**× faster than SST and SST-Center, respectively. It also achieves better latency-accuracy trade-off (1.1× faster and 0.4% higher accuracy) compared with CenterPoint++ [96]. Remarkably, FlatFormer requires 1.7× more MACs than CenterPoint++ while it is still faster. This indicates that our design is more hardware-friendly than the sparse convolutional baselines.

Deployment. We deploy our FlatFormer on an NVIDIA Jetson AGX Orin. This is a resource-constrained edge GPU platform that is widely used in real-world self-driving cars. From Figure 6, FlatFormer runs at 16 FPS, which is **1.2**× faster than CenterPoint [94] and **3**× faster than SST-Center. To the best of our knowledge, FlatFormer is the first point

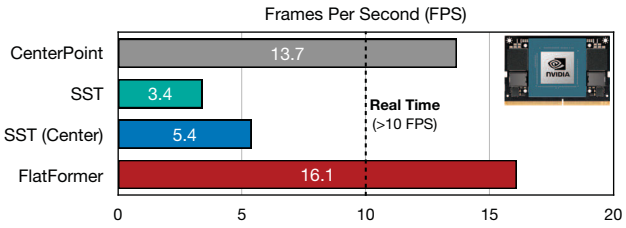

Figure 6. Measured latency on NVIDIA Jetson AGX Orin. FlatFormer is the first point cloud transformer that achieves real-time performance on edge GPUs.

cloud transformer that achieves real-time inference (*i.e.*, >10 FPS, which is the LiDAR sensor frequency) on edge GPUs. We believe that it paves the way for efficient LiDAR-centric perception in real-world applications.

5.2.2 Two-Stage Detectors

Model. To verify the generalizability, we replace the 3D backbone in FSD [20], a state-of-the-art two-stage detector, and compare its results with previous two-stage models. We keep the same grid resolution, window shape and group size as in our single-stage experiments.

Baseline & Latency. We compare our model against state-of-the-art two-stage detectors in Table 2. We follow the same latency measurement protocol. For CenterFormer [102], we adapt the official implementation to support SpConv v2.2.3 backend in FP16 precision for a fair comparison.

Results. All existing high-performing two-stage detectors are sparse convolutional, while our FlatFormer is the only transformer-based method that achieves state-of-the-art level accuracy. It also shows better scalability with respect to the number of input frames compared with CenterFormer [102]. Note that our paper focuses on optimizing the latency of *3D backbone*. However, two-stage detectors [20, 102] are usually bottlenecked by *the second stage* in runtime, which is out of our scope. We expect that the latency of FlatFormer could benefit from a more efficient second-stage design.

5.3. Analysis

In this section, we present analyses to validate the effectiveness of our design choices. All experiments are based on our single-frame model trained with 20% data.

5.3.1 Flattened Window Attention

In Figure 7, we visualize the learned attention weights in our FWA. The color represents the scale of attention weights, where warmer color means larger attention weights. Black points correspond to query points, and gray points are those with weights smaller than a threshold. For vehicles moving straight ahead, turning and parked, query points on the vehicle are always highly attended to nearby points on the same car, while faraway points have very small learned attention weights. Such an observation can partially explain the effectiveness of FWA: *i.e.*, even if equal-size grouping does not create spatially regular windows, the model can learn to suppress the importance of outlier points in the background and focus on important foreground points within each group.

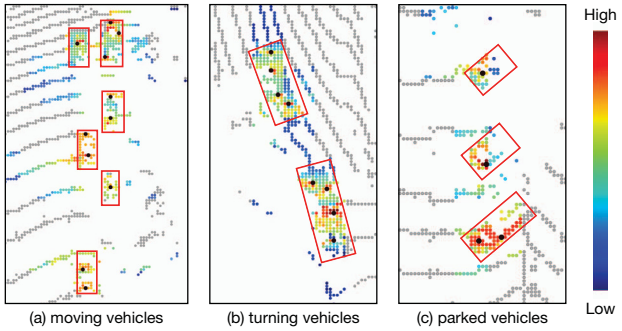

Figure 7. Visualization of attention weights in FlatFormer for vehicles that are moving straight ahead, turning and parking. High attention weights corresponds to the detected object.

Sorting Strategy	Mean L2 (mAPH)	Veh. L2 (mAPH)	Ped. L2 (mAPH)	Cyc. L2 (mAPH)
Ours	**61.7**	**63.3**	**57.9**	**63.9**
w/o Quantization	60.4	61.9	57.6	61.7
w/o Axis Alternation	61.1	63.1	57.3	62.9
w/o Window Shift	61.2	62.9	57.8	62.8
Random Order	57.8	58.8	55.1	59.4
SST [19]	60.7	62.3	56.7	63.1

Table 3. Window-based sorting in FlatFormer provides even better performance than equally-shaped window partition in SST [19] and outperforms other sorting strategies.

5.3.2 Ablation Studies on Model Design

Sorting Strategy. We first analyze the effectiveness of our window-based, axis-alternating sorting strategy in Table 3. Randomly grouping points together without any spatial sorting will give $\sim 4\%$ worse performance compared with FlatFormer. Furthermore, due to spatial discontinuities on the boundary regions, directly sorting the points by xyz coordinates or window sorting along a single axis both provide sub-optimal results. We also notice that window shifting brings about 0.5% improvement to the final performance. Interestingly, despite the fact that our sorting strategy does not guarantee the windows to be geometrically regular as in SST [19], FlatFormer still consistently outperforms SST in all three classes.

Window Shape	Mean L2 (mAPH)	Veh. L2 (mAPH)	Ped. L2 (mAPH)	Cyc. L2 (mAPH)
6×6	61.1	62.9	57.0	63.4
9×9	**61.7**	**63.3**	**57.9**	**63.9**
13×13	61.3	**63.3**	**57.9**	62.9

Table 4. FlatFormer is not sensitive to the choice of window shapes.

Group Size	Mean L2 (mAPH)	Veh. L2 (mAPH)	Ped. L2 (mAPH)	Cyc. L2 (mAPH)
81×50%	60.7	62.7	56.7	62.7
81×85%	**61.7**	**63.3**	**57.9**	**63.9**
81×125%	60.9	63.1	57.0	62.5

Table 5. Choosing a group size that is slightly smaller than the window shape (9×9) provides the best accuracy on Waymo.

Grid Resolution	Mean L2 (mAPH)	Veh. L2 (mAPH)	Ped. L2 (mAPH)	Cyc. L2 (mAPH)
0.36m	60.7	63.0	56.7	62.4
0.32m	**61.7**	**63.3**	**57.9**	63.9
0.28m	**61.7**	63.1	57.8	**64.1**

Table 6. Ablation on input resolution in FlatFormer: 0.32m×0.32m is the best design choice that balances efficiency and accuracy.

Window Shape. FlatFormer achieves robust performance under different window shapes. We choose the window shape of 9×9 (2.88m×2.88m, which is the size of a vehicle) in all experiments according to the results in Table 4, where we always fix the group size to be 85% of the window shape.

Group Size. We further study the choice of group sizes in Table 5. We fix the window shape to be 9×9 according to the results in Table 4 and sweep the group size in 50%, 85% and 125% of the window shape. The results show that setting group size to be 85% of the window shape gives the best performance. Intuitively, if the group size is too small, FlatFormer will not be able to have a large enough receptive field (*e.g.*, group size = 1, FWA will degenerate to MLP). When the group size is too large (say, group size = the entire point cloud), there will be a large number of outliers within each group, and FlatFormer will behave like a global PCT, which is not desired.

Input Resolution. From Table 6, 0.32m×0.32m input resolution is the sweet spot in the latency-accuracy tradeoff for FlatFormer while further increasing the input size will only hurt the efficiency with no performance improvements.

5.3.3 Breakdown for System Optimizations

In Figure 8, we analyze the effectiveness of our system optimizations proposed in Section 4.3. An efficient MHSA implementation (FlashAttention) brings 1.7× improvement to our inference latency. Packing the computation for $\mathcal{Q}, \mathcal{K}, \mathcal{V}$ in a single linear kernel results in a 1.3× speedup. Fusing the linear and activation layers (in FFN) brings another 1.2× speedup. Finally, dropping the non-full window improves our inference latency by 1.1×. To sum up, our system optimizations improve the latency of our FlatFormer by **2.9**×, making its backbone **2.3**× **faster** than CenterPoint [94].

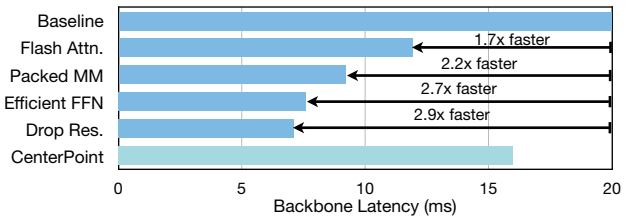

Figure 8. Improvement breakdown for system optimizations. We accelerate the backbone latency of FlatFormer by 2.9×, making it **2.3**× faster than CenterPoint.

Discussions. CenterPoint is backed by SpConv [91], which is a highly-optimized sparse convolution inference library built upon CUTLASS [31]. Nevertheless, FlatFormer still achieves the best efficiency on NVIDIA GPUs. We partially attribute our efficiency advantage to the equally-sized groups in FlatFormer which not only gives us the best computation regularity but also eliminates the computation overhead. SpConv, on the other hand, implements 3D sparse convolution with a masked implicit GEMM algorithm, which inevitably introduces computation overhead when points within one thread block do not have exactly the same neighbor patterns. As such, FlatFormer can beat sparse convolutional models on GPUs despite their heavy system optimizations.

6. Conclusion

This paper introduces FlatFormer to bridge the huge efficiency gap between point cloud transformers and sparse convolutional models. It partitions the point cloud with equal-size grouping rather than equal-window grouping, trading spatial proximity for computational regularity. FlatFormer achieves state-of-the-art accuracy on Waymo Open Dataset with 4.6× speedup over previous point cloud transformers. We hope that FlatFormer can inspire future research on designing efficient and accurate point cloud transformers.

Acknowledgement. We would like to thank Tianwei Yin, Lue Fan and Ligeng Mao for providing detailed results of CenterPoint, SST/FSD and VoTr, and Yue Wang and Yukang Chen for their helpful discussions. This work was supported by National Science Foundation, MIT-IBM Watson AI Lab, NVIDIA, Hyundai and Ford. Zhijian Liu was partially supported by the Qualcomm Innovation Fellowship.

References

[1] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. TransFusion: Robust LiDAR-Camera Fusion for 3D Object Detection with Transformers. In *CVPR*, 2022. 2

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models are Few-Shot Learners. In *NeurIPS*, 2020. 1

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *CVPR*, 2020. 2

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. In *ECCV*, 2020. 2

[5] Qi Chen, Lin Sun, Zhixin Wang, Kui Jia, and Alan Yuille. Object as Hotspots: An Anchor-Free 3D Object Detection Approach via Firing of Hotspots. In *ECCV*, 2020. 2

[6] Qi Chen, Sourabh Vora, and Oscar Beijbom. PolarStream: Streaming Lidar Object Detection and Segmentation with Polar Pillars. In *NeurIPS*, 2021. 2

[7] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-View 3D Object Detection Network for Autonomous Driving. In *CVPR*, 2017. 2

[8] Xuesong Chen, Shaoshuai Shi, Benjin Zhu, Ka Chun Cheung, Hang Xu, and Hongsheng Li. MPPNet: Multi-Frame Feature Intertwining with Proxy Points for 3D Temporal Object Detection. In *ECCV*, 2022. 2, 6

[9] Xuanyao Chen, Tianyuan Zhang, Yue Wang, Yilun Wang, and Hang Zhao. FUTR3D: A Unified Sensor Fusion Framework for 3D Detection. *arXiv*, 2022. 2

[10] Yukang Chen, Yanwei Li, Xiangyu Zhang, Jian Sun, and Jiaya Jia. Focal Sparse Convolutional Networks for 3D Object Detection. In *CVPR*, 2022. 2

[11] Yukang Chen, Jianhui Liu, Xiaojuan Qi, Xiangyu Zhang, Jian Sun, and Jiaya Jia. Scaling up Kernels in 3D CNNs. *arXiv*, 2022. 2

[12] Yilun Chen, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast Point R-CNN. In *ICCV*, 2019. 2

[13] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *NeurIPS*, 2021. 2

[14] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *CVPR*, 2019. 1, 2

[15] Ozgun Cicek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. In *MICCAI*, 2016. 2

[16] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *NeurIPS*, 2022. 5

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 2019. 1, 2

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021. 1

[19] Lue Fan, Ziqi Pang, Tianyuan Zhang, Yu-Xiong Wang, Hang Zhao, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. Embracing Single Stride 3D Object Detector with Sparse Transformer. In *CVPR*, 2022. 1, 2, 3, 4, 5, 6, 7

[20] Lue Fan, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. Fully Sparse 3D Object Detection. In *NeurIPS*, 2022. 2, 6, 7

[21] Lue Fan, Xuan Xiong, Feng Wang, Naiyan Wang, and ZhaoXiang Zhang. RangeDet: In Defense of Range View for LiDAR-Based 3D Object Detection. In *ICCV*, 2021. 2

[22] Runzhou Ge, Zhuangzhuang Ding, Yihan Hu, Wenxin Shao, Li Huang, Kun Li, and Qiang Liu. 1st Place Solutions to the Real-time 3D Detection and the Most Efficient Model of the Waymo Open Dataset Challenge 2021. In *CVPRW*, 2021. 2

[23] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *IJRR*, 2013. 2

[24] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D Semantic Segmentation With Submanifold Sparse Convolutional Networks. In *CVPR*, 2018. 1, 2

[25] Tianrui Guan, Jun Wang, Shiyi Lan, Rohan Chandra, Zuxuan Wu, Larry Davis, and Dinesh Manocha. M3DETR: Multi-Representation, Multi-Scale, Mutual-Relation 3D Object Detection With Transformers. In *WACV*, 2022. 2

[26] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. PCT: Point Cloud Transformer. *CVM*, 2021. 1, 2, 3

[27] Chenhang He, Ruihuang Li, Shuai Li, and Lei Zhang. Voxel Set Transformer: A Set-to-Set Approach to 3D Object Detection from Point Clouds. In *CVPR*, 2022. 2, 5, 6

[28] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. In *CVPR*, 2020. 2

[29] Yihan Hu, Zhuangzhuang Ding, Runzhou Ge, Wenxin Shao, Li Huang, Kun Li, and Qiang Liu. AFDetV2: Rethinking the Necessity of the Second Stage for Object Detection from Point Clouds. In *AAAI*, 2022. 2

[30] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention. In *ICML*, 2020. 5

[31] Andrew Kerr, Haicheng Wu, Manish Gupta, Dustyn Blasig, Pradeep Ramini, et al. CUTLASS, 2022. 8

[32] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in Vision: A Survey. *ACM Comput. Surv.*, 2022. 2

[33] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021. 2, 3

[34] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, and Jiong Yang. PointPillars: Fast Encoders for Object Detection from Point Clouds. In *CVPR*, 2019. 2, 5, 6

[35] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on $\mathcal{X}$-Transformed Points. In *NeurIPS*, 2018. 2

[36] Yanwei Li, Yilun Chen, Xiaojuan Qi, Zeming Li, Jian Sun, and Jiaya Jia. Unifying Voxel-based Representation with Transformer for 3D Object Detection. In *NeurIPS*, 2022. 2

[37] Yanghao Li, Hanzi Mao, Girshick, and Kaiming He. Exploring Plain Vision Transformer Backbones for Object Detection. In *ECCV*, 2022. 2

[38] Yingwei Li, Adams Wei Yu, Tianjian Meng, Ben Caine, Jiquan Ngiam, Daiyi Peng, Junyang Shen, Bo Wu, Yifeng Lu, Denny Zhou, et al. DeepFusion: Lidar-Camera Deep Fusion for Multi-Modal 3D Object Detection. In *CVPR*, 2022. 2

[39] Zhichao Li, Feng Wang, and Naiyan Wang. LiDAR R-CNN: An Efficient and Universal 3D Object Detector. In *CVPR*, 2021. 2, 6

[40] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Yu Qiao, and Jifeng Dai. BEVFormer: Learning Bird's-Eye-View Representation from Multi-Camera Images via Spatiotemporal Transformers. In *ECCV*, 2022. 2

[41] Zhiqi Li, Wenhai Wang, Enze Xie, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, Tong Lu, and Ping Luo. Panoptic SegFormer: Delving Deeper into Panoptic Segmentation with Transformers. In *CVPR*, 2021. 2

[42] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep Continuous Fusion for Multi-Sensor 3D Object Detection. In *ECCV*, 2018. 2

[43] Zhijian Liu, Alexander Amini, Sibo Zhu, Sertac Karaman, Song Han, and Daniela Rus. Efficient and Robust LiDAR-Based End-to-End Navigation. In *ICRA*, 2021. 2

[44] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin Transformer V2: Scaling Up Capacity and Resolution. In *CVPR*, 2022. 2, 3, 5

[45] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In *ICCV*, 2021. 1, 2, 3, 5

[46] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han. BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation. In *ICRA*, 2023. 2

[47] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-Voxel CNN for Efficient 3D Deep Learning. In *NeurIPS*, 2019. 1, 2

[48] Zhijian Liu, Haotian Tang, Shengyu Zhao, Kevin Shao, and Song Han. PVNAS: 3D Neural Architecture Search with Point-Voxel Convolution. *TPAMI*, 2021. 2

[49] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated Convolutional Networks for 3D Point Cloud Understanding. In *ICCV*, 2019. 2

[50] Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu. Voxel Transformer for 3D Object Detection. In *ICCV*, 2021. 1, 2, 3, 5, 6

[51] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *IROS*, 2015. 2

[52] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3D Object Detection with Pointformer. In *CVPR*, 2021. 2, 3

[53] Chunghyun Park, Yoonwoo Jeong, Minsu Cho, and Jaesik Park. Fast Point Transformer. In *CVPR*, 2022. 2, 3

[54] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. In *CVPR*, 2018. 2

[55] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*, 2017. 2

[56] Charles Ruizhongtai Qi, Hao Su, Matthias Niessner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and Multi-View CNNs for Object Classification on 3D Data. In *CVPR*, 2016. 2

[57] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NeurIPS*, 2017. 1, 2

[58] Charles R Qi, Yin Zhou, Mahyar Najibi, Pei Sun, Khoa Vo, Boyang Deng, and Dragomir Anguelov. Offboard 3d object detection from point cloud sequences. In *CVPR*, 2021. 2

[59] Guocheng Qian, Hasan Abed Al Kader Hammoud, Guohao Li, Ali Thabet, and Bernard Ghanem. ASSANet: An Anisotropical Separable Set Abstraction for Efficient Point Cloud Representation Learning. In *NeurIPS*, 2021. 2

[60] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *CVPR*, 2017. 2

[61] B. Helou S. Vora, A. H. Lang and O. Beijbom. PointPainting: Sequential Fusion for 3D Object Detection. In *CVPR*, 2020. 2

[62] Guangsheng Shi, Ruifeng Li, and Chao Ma. PillarNet: Real-Time and High-Performance Pillar-based 3D Object Detection. In *ECCV*, 2022. 2, 5, 6

[63] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection. In *CVPR*, 2020. 2, 6

[64] Shaoshuai Shi, Li Jiang, Jiajun Deng, Zhe Wang, Chaoxu Guo, Jianping Shi, Xiaogang Wang, and Hongsheng Li. PV-RCNN++: Point-Voxel Feature Set Abstraction With Local Vector Representation for 3D Object Detection. *arXiv*, 2021. 2, 6

[65] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In *CVPR*, 2019. 2

[66] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network. *TPAMI*, 2020. 2

[67] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network. *TPAMI*, 2020. 6

[68] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *ICCV*, 2017. 2

[69] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In *CVPR*, 2020. 2, 3, 5

[70] Pei Sun, Mingxing Tan, Weiyue Wang, Chenxi Liu, Fei Xia, Zhaoqi Leng, and Dragomir Anguelov. SWFormer: Sparse Window Transformer for 3D Object Detection in Point Clouds. In *ECCV*, 2022. 1, 2

[71] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. TorchSparse: Efficient Point Cloud Inference Engine. In *MLSys*, 2022. 1

[72] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. In *ECCV*, 2020. 2

[73] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *ICCV*, 2019. 2

[74] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training Data-Efficient Image Transformers & Distillation through Attention. In *ICML*, 2021. 1, 2

[75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *NeurIPS*, 2017. 1, 2, 3, 4

[76] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ToG*, 2017. 2

[77] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes. In *SIGGRAPH Asia*, 2018. 2

[78] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. In *ICCV*, 2021. 2

[79] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. PVTv2: Improved Baselines with Pyramid Vision Transformer. *CVM*, 2022. 2

[80] Wenxiao Wang, Lu Yao, Long Chen, Binbin Lin, Deng Cai, Xiaofei He, and Wei Liu. CrossFormer: A Versatile Vision Transformer Hinging on Cross-scale Attention. In *ICLR*, 2022. 2

[81] Yue Wang, Vitor Guizilini, Tianyuan Zhang, Yilun Wang, Hang Zhao, , and Justin M. Solomon. DETR3D: 3D Object Detection from Multi-view Images via 3D-to-2D Queries. In *CoRL*, 2021. 2

[82] Yue Wang and Justin M. Solomon. Object DGCNN: 3D Object Detection using Dynamic Graphs. In *NeurIPS*, 2021. 2

[83] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ToG*, 2019. 2

[84] Zhixin Wang and Kui Jia. Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection. In *IROS*, 2019. 2

[85] Wenxuan Wu, Zhongang Qi, and Li Fuxin. PointConv: Deep Convolutional Networks on 3D Point Clouds. In *CVPR*, 2019. 2

[86] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point Transformer V2: Grouped Vector Attention and Partition-based Pooling. In *NeurIPS*, 2022. 2, 3

[87] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *CVPR*, 2015. 2

[88] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. In *NeurIPS*, 2021. 2

[89] Qiangeng Xu, Yin Zhou, Weiyue Wang, Charles R. Qi, and Dragomir Anguelov. Grid-gcn for fast and scalable point cloud learning. In *CVPR*, 2020. 2

[90] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. PointASNL: Robust Point Clouds Processing Using Nonlocal Neural Networks with Adaptive Sampling. In *CVPR*, 2020. 2, 3

[91] Yan Yan, Yuxing Mao, and Bo Li. SECOND: Sparsely Embedded Convolutional Detection. *Sensors*, 2018. 1, 2, 5, 6, 8

[92] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3DSSD: Point-based 3D Single Stage Object Detector. *CVPR*, 2020. 2

[93] Dongqiangzi Ye, Weijia Chen, Zixiang Zhou, Yufei Xie, Yu Wang, Panqu Wang, and Hassan Foroosh. LidarMultiNet: Unifying LiDAR Semantic Segmentation, 3D Object Detection, and Panoptic Segmentation in a Single Multi-task Network. *arXiv*, 2022. 2

[94] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3D Object Detection and Tracking. In *CVPR*, 2021. 1, 2, 3, 5, 6, 8

[95] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Multimodal Virtual Point 3D Detection. In *NeurIPS*, 2021. 2

[96] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-Point++ Submission to the Waymo Real-time 3D Detection Challenge. Technical report, 2022. 2, 5, 6

[97] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis E.H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-Token ViT: Training Vision Transformers From Scratch on ImageNet. In *ICCV*, 2021. 2

[98] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point Transformer. In *ICCV*, 2021. 1, 2, 3

[99] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip H.S. Torr, and Li Zhang. Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers. In *CVPR*, 2021. 2

[100] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-End Multi-View Fusion for 3D Object Detection in LiDAR Point Clouds. *CoRL*, 2019. 2

[101] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *CVPR*, 2018. 2

[102] Zixiang Zhou, Xiangchen Zhao, Yu Wang, Panqu Wang, and Hassan Foroosh. CenterFormer: Center-based Transformer for 3D Object Detection. In *ECCV*, 2022. 2, 6, 7

[103] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-Balanced Grouping and Sampling for Point Cloud 3D Object Detection. *arXiv*, 2019. 2

[104] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable Transformers for End-to-End Object Detection. In *ICLR*, 2021. 2