APTO: Accelerating Serialization-Based Point Cloud Transformers with Position-Aware Pruning

Qichu Sun*
sunqichu22z@ict.ac.cn
State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Rui Meng*
mengrui22s@ict.ac.cn
State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Haishuang Fan
fanhaishuang20z@ict.ac.cn
State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Fangqiang Ding
f.ding-1@sms.ed.ac.uk
School of Informatics, University of Edinburgh
Edinburgh, UK

Linxi Lu
lulinxi23z@ict.ac.cn
State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Jingya Wu
wujingya@ict.ac.cn
State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China

Xiaowei Li
lxw@ict.ac.cn
State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China

Guihai Yan
yan@ict.ac.cn
State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences
YUSUR Technology Co., Ltd.
Beijing, China

ABSTRACT

Point cloud processing has broad applications in autonomous driving and robotics. Serialization-based point cloud transformers map unordered point clouds onto directed curves, use sparse convolution for down-sampling and apply attention in local windows to capture spatial relationships. Despite achieving great accuracy, these models face inference latency challenges: neighbor search in sparse convolution exhibits low parallelism; attention computation remains complex, especially with larger window sizes; softmax introduces data dependencies. This paper proposes APTO, an accelerator for serialization-based models. It uses voxels' z-curve indices to perform neighbor searches in parallel, employs a position-aware pruning strategy using neighboring voxel counts to eliminate useless attention computations, and adopts a fine-grained attention dataflow for parallel processes with minimal data dependencies. Besides, its hardware has dedicated computation cores for efficient processing. Evaluations show that APTO achieves average 10.22×, 3.53× and 2.70× speedups over RTX 4090 GPU, PointAcc, and SpOctA, with 153.59×, 8.57× and 7.25× energy savings.

CCS CONCEPTS

• **Hardware** → *Application specific processors*; *Hardware accelerators*; *Hardware-software codesign.*

KEYWORDS

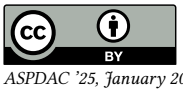
Point Cloud, Transformer, Serialization, Attention, Accelerator

ACM Reference Format:
Qichu Sun, Rui Meng, Haishuang Fan, Fangqiang Ding, Linxi Lu, Jingya Wu, Xiaowei Li, and Guihai Yan. 2025. APTO: Accelerating Serialization-Based Point Cloud Transformers with Position-Aware Pruning. In *30th Asia and South Pacific Design Automation Conference (ASPDAC '25), January 20–23, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3658617.3697655

1 INTRODUCTION

Point cloud is a popular representation of 3D data, obtained from LiDARs or RGB-D cameras, containing valuable geometry and color information. Processing point clouds enables tasks like object detection [29], classification [17], and segmentation [30], and finds wide application in autonomous driving, robotic perception, and AR/VR. As the usage grows, achieving reliable, accurate, real-time, and energy-efficient point cloud processing becomes crucial.

*Both authors contributed equally to this research.



ASPDAC '25, January 20–23, 2025, Tokyo, Japan

ACM ISBN 979-8-4007-0635-6/25/01
https://doi.org/10.1145/3658617.3697655

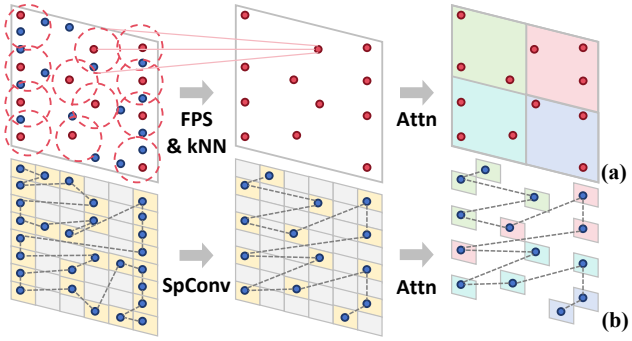


Figure 1: (a) Point-based model. (b) Serialization-based model. SpConv replaces FPS and kNN for down-sampling, and attention windows are divided based on indices on the curve.

As point clouds expand in scale and processing tasks grow in complexity, new models are emerging. The point-based method [7, 27] integrates transformer [20] into point cloud processing, using Farthest Point Sampling (FPS) and k-Nearest Neighbors (kNN) for down-sampling, and attention mechanisms in local windows for feature computation (as shown in Fig. 1 (a)). However, its reliance on FPS and kNN involves substantial repetitive point access, resulting in inefficient memory utilization and computational redundancy, and the limited window size restricts its accuracy.

To address the inefficiency of down-sampling in point-based models and capture large-range relationships in point clouds, the serialization-based method [11, 22, 23] has been proposed. As shown in Fig. 1 (b), this method first restructures point clouds into tiny voxels and organizes them onto a non-intersecting curve (e.g. z-curve [15]), which converts point clouds into ordered sequences, enabling regular memory access. Then it uses 3D sparse convolution (SpConv) [5] to replace FPS and kNN for efficient down-sampling. SpConv involves fewer redundant computations, preserves more important features in down-sampling, and is more effective for large-scale point clouds. Finally, it partitions the down-sampled voxels into windows based on their orders on the curve and applies attention mechanisms in these windows to extract voxel features. It usually adopts larger attention windows to capture long-range relations and has better accuracy on diverse point cloud datasets.

Although serialization-based models have achieved great accuracy in large-scale point clouds, high latency and memory usage hinder their effective deployment on resource-constrained general platforms. For example, Octformer [22] exhibits an 85ms latency per frame on an RTX 4090 GPU for segmentation tasks. Hence, hardware acceleration becomes critical for developing high-throughput point cloud systems. However, prior point cloud accelerators [4, 6, 9, 10, 18, 28] only focus on accelerating FPS and kNN-based models, and accelerating serialization-based models remains a challenge. Upon analyzing the serialization-based method, we find it faces three challenges: 1) **Inefficient SpConv Operations.** SpConv can be divided into two stages: mapping and aggregating. In the mapping stage, neighbor search connects input and output voxels and weights in the kernel map. Prior methods [10, 25] performed neighbor searches serially by shifting the input voxels by one position and merging all input and output voxels to find voxels at the same location. Considering the massive number of voxels in serialization-based models, this approach is inefficient and lacks parallelism. 2) **Useless Attention Computation.** Serialization-based models use window attention to reduce the complexity of classical transformers. However, not all voxels in the attention window are important, and the unimportant ones introduce significant useless computation. Previous works [12, 21, 26] use static or dynamic pruning methods but introduce non-trivial accuracy loss or significant computational overhead. Designing lightweight pruning strategies to avoid useless computation in the point cloud process remains crucial. 3) **Data Dependencies in Softmax.** To avoid numerical overflow, softmax subtracts the maximum value (m_i) of each row from the score matrix before exponentiation. Since m_i is involved in all multiplications between query and key of all tokens, attention has severe data dependencies and hinders efficient pipeline for different tokens' computations.

To address these challenges, we propose a customized accelerator APTO that reduces the complexity of serialization-based models' attention computation. First, we observe that during neighbor searches, the shifted input and output voxels at the same location have the same z-curve indices, and voxels can be divided into 8 non-overlapping sets based on the lowest 3 bits ϕ_0 of their z-curve indices. Hence, we only need to search for voxels at the same location within the same set. Leveraging this, APTO divides voxels into 8 sets based on ϕ_0 and performs parallel neighbor searches for different sets to accelerate the mapping stage in SpConv. Second, voxels in point cloud space can be divided into boundary voxels with fewer neighbors and interior voxels with more neighbors, and the former is more important than the latter. We find that in attention computation, tokens from voxels with fewer neighbors carry more weight, and pruning tokens from voxels with more neighbors has minimal impact on the results. Based on this observation, APTO adopts a position-aware pruning strategy. It identifies neighboring voxel counts and classifies voxels into boundary and interior voxels in SpConv, minimizing the preprocessing overhead. In attention computation, APTO directly skips the token computation of interior voxels and removes numerous useless computations. Third, APTO replaces the maximum value of a row in softmax with the dynamic maximum value of calculated elements to reduce data dependencies, and partitions **K** and **V** matrices into blocks to gain parallelism along the sequence dimension. Moreover, APTO incorporates a hardware design with three dedicated computing cores. Our contributions are summarized as follows:

- APTO optimizes SpConv's mapping stage by searching neighbor voxels in parallel based on the lowest 3 bits of their z-curve indices.
- APTO takes a position-aware attention pruning strategy to reduce redundant attention computations, which prunes tokens generated from voxels with more neighbors.
- APTO adopts a fine-grained attention dataflow with dynamic maximum values to remove data dependencies and splits **K-V** into blocks for parallel processing.

Experimental results show that APTO achieves average 10.22×, 3.53× and 2.70× speedups over RTX 4090 GPU, PointAcc [10], and SpOctA [13], with 153.59×, 8.57× and 7.25× energy savings, enabling efficient processing for large-scale point clouds on resource-constrained devices. To the best of our knowledge, APTO is the first work to accelerate serialization-based point cloud transformers.

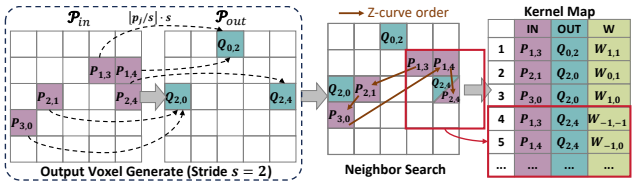


Figure 2: Mapping stage of SpConv. Unlike image convolutions, the output coordinates are predefined.

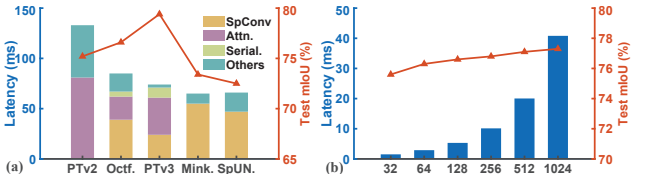


Figure 3: (a) Performance of models on ScanNet. (b) Attention latency and test mIoU of PTv3 as window size increases.

2 PRELIMINARIES

2.1 Transformer in Point Clouds

Recently transformer [20] has seen widespread adoption in various scenarios, including point cloud processing. In point clouds, self-attention is calculated with points in local windows:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\mathbf{Q}\mathbf{K}^T/\sqrt{d_k}\right)\mathbf{V} \tag{1}$$

where $\mathbf{Q}_{n\times d_k}$, $\mathbf{K}_{n\times d_k}$, and $\mathbf{V}_{n\times d_k}$ represent query, key, and value matrices, obtained through matrix multiplication of input point features with their weight matrices, and the result of $\mathbf{Q}\mathbf{K}^T$ is the score matrix $\mathbf{S}$. Furthermore, positional encoding is carried out before self-attention to add position information.

2.2 3D Sparse Convolution

Image convolution is inefficient for point clouds due to their sparsity. However, 3D sparse convolution (SpConv) [5] offers efficient processing without sacrificing accuracy and is compatible with sparse data storage. This makes it a preferred choice for point cloud algorithms [2, 5]. Specifically, SpConv inputs voxel coordinates and features, and generates output coordinates $q_i = \lfloor p_j/s \rfloor \cdot s$ from input coordinates p_j with a stride s. Next, the output voxels traverse neighboring positions, accumulating the products of non-zero input voxel features and weights to form output voxel features. Generally, SpConv's usage is determined by s: $s = 1$ for operations with identical input and output coordinates like positional encoding; $s > 1$ for down-sampling. Typically, SpConv involves two stages: 1) **Mapping.** As shown in Fig. 2, a kernel map establishes relationships among input and output voxels and weights. Each output voxel iterates through its neighbors to find corresponding input voxels and store entries in the kernel map. 2) **Aggregating.** Based on the kernel map, features of input voxels corresponding to the same output voxel are multiplied by weights and accumulated.

2.3 Serialization-Based Models

To address the inefficiency of down-sampling and capture point relationships over a larger range, serialization-based models have been developed. These models transform unstructured point clouds into ordered sequences by mapping point coordinates onto a non-intersecting curve, such as the z-curve [15]. Specifically, the z-curve method recursively subdivides a spatial cube into 8 smaller cubes until they reach a preset voxel size or depth threshold. For a voxel with coordinates $\theta = (x, y, z)$, where x, y, z are in binary as $\{x_{n-1}...x_1x_0\}_2$, $\{y_{n-1}...y_1y_0\}_2$, $\{z_{n-1}...z_1z_0\}_2$, its z-curve index is:

$$\Phi = \{\phi_{n-1}...\phi_1\phi_0\}_8 = \{z_{n-1}y_{n-1}x_{n-1}...z_1y_1x_1z_0y_0x_0\}_2 \tag{2}$$

For instance, a voxel with coordinates $\theta = (x, y, z) = (6, 3, 1)$ where $x = 110_2$, $y = 011_2$, $z = 001_2$ has a z-curve index $\Phi = 463_8 = 100110011_2$ formed by combining these binaries bit-wise.

Compared to FPS and kNN, serialization-based models employ more efficient operations like SpConv for down-sampling, and partition attention windows based on voxels' orders on the curve. Additionally, positional encoding before attention is also applied with SpConv (as shown in Fig. 4 (b)). Among these models, Octformer [22] incorporates an octree to map points onto a z-curve, using octree-based SpConv for down-sampling; Point Transformer v3 (PTv3) [23] serializes point clouds with diverse patterns including z-curve, and performs attention computation in large-size windows, expanding its receptive field from 16 to 1024 points. PTv3 achieves state-of-the-art performance on various large-scale datasets.

3 MOTIVATION

Serialization-based models excel in large-scale point cloud applications but encounter computational complexity challenges, limiting their practical utility. For instance, in segmentation tasks on ScanNet [3] with an average of 100k points per frame, Octformer exhibits an 85ms latency on an RTX 4090 GPU, and PTv3 has a latency of 74ms (as shown in Fig. 3 (a)). In contrast, traditional models like MinkUNet [2] and SpUNET [5] achieve lower latencies of 65ms and 66ms on the same GPU. In the inference process of Octformer, SpConv contributes 46% of total time, with attention accounting for 27%, whereas in PTv3, they are 32% and 50%, underscoring their roles as bottlenecks. This inefficiency is attributed to three factors.

3.1 Inefficient SpConv Operations

SpConv is a bottleneck in serialization-based models, as shown in Fig. 3 (a), due to the high computation complexity in the mapping stage. In this stage, the previous method [10, 25] relies on voxel merging to find input and output voxels connected by the same weight in the kernel map. This involves shifting input voxels in a specific direction (e.g., (1, 1) for $W_{-1,-1}$ in Fig. 4 (a)), merging shifted input voxels with output voxels, and sequentially searching for voxels at the same location to add corresponding $(input, output, weight)$ entries to the kernel map. However, it merges and compares all voxels serially, resulting in unnecessary operations and lacking parallelism. Therefore, this method is inefficient considering serialization-based models' large voxel quantity (over 10^5). Besides, an alternative approach [13] uses a neighbor lookup table for parallel neighbor searches but results in substantial on-chip resource overhead.

3.2 Useless Attention Computation

Serialization-based models use attention in local windows to capture spatial relationships effectively. Within a window, each voxel engages in attention calculations with every other voxel, though not all voxels are critical. The unimportant voxels introduce a large amount of useless computation. To eliminate useless calculations in the attention mechanism, pruning is commonly used, which can

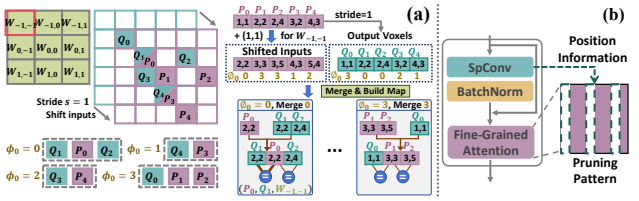


Figure 4: (a) Parallel neighbor search. (b) Attention pruning.

be categorized into dynamic and static methods. On the one hand, the dynamic method [12, 21] pre-calculates a low-precision score matrix to guide which token can be pruned, introducing additional computation and storage overhead. On the other hand, the static method [26] adopts fixed pruning patterns, causing significant accuracy loss. In summary, both methods cannot suit serialization-based models, and a dedicated pruning scheme is needed to reduce useless attention computation, minimizing accuracy loss without introducing additional computation. Additionally, although increasing the window size improves model accuracy, it also results in higher attention latency (as shown in Fig. 3 (b)). With a fixed point cloud size of N and a window size of w, the overall complexity of attention calculations is $O(Nw)$, which grows rapidly as w increases.

3.3 Data Dependencies in Softmax

During attention computation, the softmax operator poses a bottleneck due to exponential functions risking numerical overflow with large input data, potentially leading to accuracy loss [14]. In existing attention algorithms, it's common practice to subtract the maximum value from each row in the score matrix $\mathbf{S}$ before performing exponential operations to prevent overflow. However, this approach introduces significant data dependencies. Specifically, each query vector $\mathbf{q}_i$ are computed with all key vectors $\mathbf{k}_j$ and value vectors $\mathbf{v}_j$ for the output, which can be expressed as:

$$output_i = \text{softmax}(\mathbf{S}_i) \cdot \mathbf{V} = \frac{\sum_{j=1}^{n} exp(s_{ij} - m_i)\mathbf{v}_j}{\sum_{j=1}^{n} exp(s_{ij} - m_i)} \tag{3}$$

where s_{ij} is a score in the i-th row of $\mathbf{S}$, derived from multiplicating $\mathbf{q}_i$ and $\mathbf{k}_j$, and m_i is the maximum s_{ij}. Since m_i can only be calculated after all $\mathbf{k}_j$ multiplications, severe data dependencies exist, inhibiting an efficient pipeline for computations of different $\mathbf{k}_j$, $\mathbf{v}_j$.

4 OPTIMIZATIONS

4.1 Serialization-based Parallel Neighbor Search

By leveraging serialization, a parallel search strategy can be used to accelerate neighbor searches in SpConv. After serialization, the point cloud space is divided into small voxels, with each non-empty voxel assigned a z-curve index. This allows for parallel neighbor searches with minimal hardware requirements. Specifically, based on the last 3 bits ϕ_0 of their z-curve indices, voxels can be organized into 8 non-overlapping sets with a well-balanced distribution.

As shown in Fig. 4 (a), input and output voxels are merged, and neighboring elements in the array are compared to add entries to the kernel map for voxels in the same position. By utilizing the z-curve indices of voxels, this process can be optimized to increase parallelism. In detail, input and output voxels are grouped based on the lowest 3 bits ϕ_0 of their z-curve indices, with each group performing merging and comparison in parallel. Fig. 4 (a) illustrates

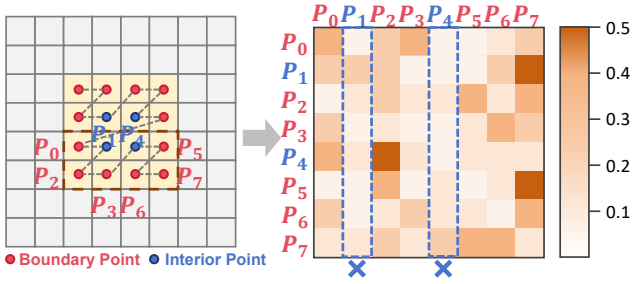


Figure 5: Position-aware attention pruning. Tokens for voxels with more neighbors are pruned to reduce redundancy.

an example in 2D, where Q_0 and the shifted P_1 share the same lowest bits $\phi_0 = 3$, placing them in the same set for processing, and different voxel sets are processed in parallel. The hardware details on this parallel neighbor search strategy are illustrated in Sec. 5.1.

4.2 Position-Aware Attention Pruning

Pruning can reduce redundant attention calculations. Unlike existing pruning methods [12, 21, 26], we use inherent positional information in the point cloud for pruning, eliminating redundant tokens in attention without extra pre-calculations. Specifically, the distribution of non-empty voxels in the point cloud space is uneven, with numerous voxels having more neighbors. Typically, voxels with more neighbors indicate being situated in the interior of objects, while those with fewer neighbors tend to represent object boundaries. In point cloud processing, object boundaries are more crucial than interiors; for instance, once an object's boundary is pinpointed, it becomes feasible to be segmented from the background. Consequently, in attention computation, tokens from voxels with fewer neighbors carry more weight, and pruning tokens from voxels with many neighbors has minimal impact on the results. Besides, before attention computation, the neighbor searching process in SpConv identifies neighboring voxel counts.

In the position-aware pruning method, we utilize results from positional encoding implemented through SpConv (as shown in Fig. 4 (b)), which embeds voxels' position information into the input for attention computation, indicating the number of neighbors each voxel has. During attention computation, as illustrated in Fig. 5, tokens linked to voxels with more neighbors (P_1 and P_4) are pruned. In other words, the key and value vectors ($\mathbf{k}_j$ and $\mathbf{v}_j$) for these voxels are no longer calculated. This pruning strategy significantly reduces the redundant computations in the attention mechanism.

4.3 Fine-Grained Attention Dataflow

In serialization-based models, subtracting the row's maximum value m_i in softmax introduces data dependencies during attention computations. If we decouple the computation of each $\mathbf{k}_j$ in $\mathbf{K}$ and use the m_i of the scores s_{ij} already calculated instead of the entire row, these data dependencies can be eliminated. Moreover, by partitioning keys and values into blocks, computations can be parallelized along the sequence dimension, thereby enhancing efficiency.

To deal with the data dependencies, we propose a fine-grained dataflow that utilizes dynamic maximum value m_i for s_{ij}, and handles attention in large windows by partitioning $\mathbf{K}$ and $\mathbf{V}$ into Y blocks along the sequence dimension, with each block computed in

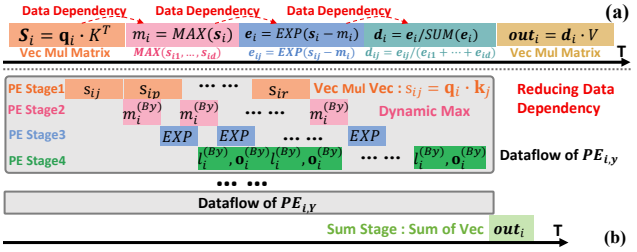


Figure 6: (a) Attention dataflow with data dependencies. (b) Fine-grained dataflow using dynamic maximum value m_i.

parallel. As shown in Fig. 6 (b), each time an s_{ij} is computed, it is exponentiated with the current m_i, eliminating the need to wait for all $\mathbf{k}_j$ multiplications to obtain the m_i of the entire row in Fig. 6 (a), reducing data dependencies. As shown in Algo. 1, the calculation for each block B_y can be divided into 4 stages, with the numerator and denominator of each block denoted as $\mathbf{o}_i$ and l_i. At stage 1, $\mathbf{q}_i$ and $\mathbf{k}_j$ are multiplied to obtain s_{ij}. At stage 2, the stored m_i is compared with s_{ij} to get a new m_i. At stage 3, exponentiation is calculated, subtracting the new m_i. At stage 4, the multiplication of s_{ij} and $\mathbf{v}_j$ is conducted, with the results updated by the difference between the stored and new m_i, and the stored m_i is updated to ensure it becomes the maximum among all s_{ij} after all iterations.

After computations of all blocks are completed, their results are aggregated. The global maximum value M_i of all local m_i is obtained. The overall attention result of $\mathbf{q}_i$, as depicted in Algo. 1, is achieved by updating $\mathbf{o}_i$ and l_i from each block with the difference between their m_i and M_i. By breaking down attention calculations in large windows, this method avoids large input and intermediate matrices and leverages parallelism in the sequence dimension. Moreover, its reconfigurable calculations for each $\mathbf{k}_j$ make it suitable for our proposed pruning strategy, avoiding low PE utilization due to workload imbalance. It also enables hardware design in Sec. 5.3.

Algorithm 1: Fine-grained Attention Mechanism

INPUT: $\mathbf{q}_1, \ldots, \mathbf{q}_n, \mathbf{k}_1, \ldots, \mathbf{k}_n$ and $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of size $1 \times d_k$ as rows of $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ of size $n \times d_k$

OUTPUT: $\mathbf{output}_1, \ldots, \mathbf{output}_n$ of size $1 \times d_k$ as rows of attention output matrix of size $n \times d_k$

Parallel for i **in** *range(n)* **do**

  Parallel for B_y **in** $\{B_1, B_2, \ldots, B_Y\}$ **do**

    Initialize $\mathbf{o}_i^{(B_y)} = (0)_{1\times d_k}$, $l_i^{(B_y)} = 0$, $m'^{(B_y)}_i = -\infty$

    for j **in** B_y **do**

      $s_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j^T$ — **Stage 1**

      $m_i^{(B_y)} = max\left(m'^{(B_y)}_i, s_{ij}\right)$ — **Stage 2**

      $\tilde{s}_{ij} = exp\left(s_{ij} - m_i^{(B_y)}\right)$

      $\tilde{m}_i^{(B_y)} = exp\left(m'^{(B_y)}_i - m_i^{(B_y)}\right)$ — **Stage 3**

      $l_i^{(B_y)} = \tilde{m}_i^{(B_y)} l_i^{(B_y)} + \tilde{s}_{ij}$

      $\mathbf{o}_i^{(B_y)} = \tilde{m}_i^{(B_y)} \cdot \mathbf{o}_i^{(B_y)} + \tilde{s}_{ij} \cdot \mathbf{v}_j$ — **Stage 4**

      $m'^{(B_y)}_i = m_i^{(B_y)}$

  $M_i = MAX\left(m_i^{(B_1)}, m_i^{(B_2)}, \ldots, m_i^{(B_Y)}\right)$

  $\mathbf{output}_i = \frac{\sum_y exp\left(m_i^{(B_y)} - M_i\right) \cdot \mathbf{o}_i^{(B_y)}}{\sum_y exp\left(m_i^{(B_y)} - M_i\right) l_i^{(B_y)}}$ — **SUM**

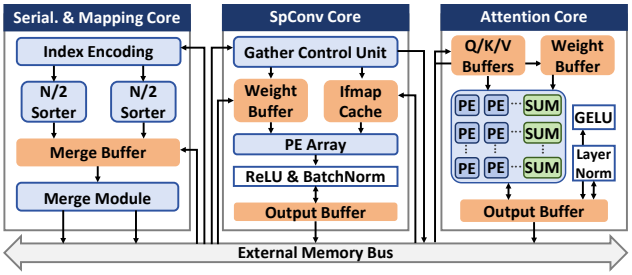


Figure 7: Overview of APTO architecture.

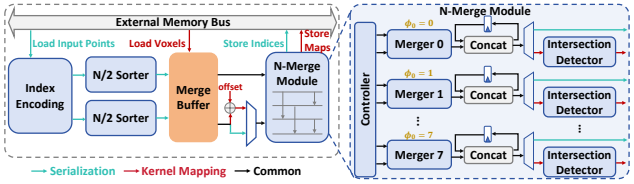


Figure 8: Serialization and mapping core. Voxels are processed in parallel by 8 mergers in the N-merge module.

5 HARDWARE DESIGN

Fig. 7 shows APTO's hardware architecture. It uses three dedicated cores to support serialization-based models. Specifically, the serialization and mapping core maps point coordinates to z-curve indices and locates neighboring voxels, and the SpConv and attention cores perform the respective computations. Data are exchanged with high-bandwidth memory (HBM) via the external memory bus.

5.1 Serialization and Mapping Core

The serialization and mapping core is responsible for both serialization and kernel map creation. For serialization, the index encoding module retrieves point coordinates from external memory and converts them into z-curve indices (see Sec. 2.3), which are then sorted. In detail, the $N/2$ sorter sorts $N/2$ indices, and the merge module combines two sorted arrays into one.

In terms of kernel map creation, this core supports parallel neighbor searching (see Sec. 4.1), where voxels are divided into 8 groups based on their z-curve indices' lowest 3 bits ϕ_0. In the N-merge module, the shifted input voxels and output voxels are merged, and the intersection detectors find voxels in the same position. As shown in Fig. 8, tasks are assigned by the controller using ϕ_0, and 8 identical merger-detector series process these tasks in parallel.

5.2 SpConv Core

The SpConv core comprises a 64×16 PE array for aggregating in SpConv and a gather control unit for data preparation. The gather control unit reads the output voxel coordinates and appends them to the ID FIFO. Simultaneously, it retrieves the corresponding kernel map entries from memory based on the ID FIFO's front element and stores them in the map buffer. As shown in Fig. 9, a counter generates a bitmap with neighboring voxel counts to guide pruning in the attention core. When entries related to an output voxel start to be loaded, the counter is reset to 0. Then it counts the total number of entries for this voxel. If the count exceeds a predefined threshold, the corresponding element in the bitmap is set to 2'b1.

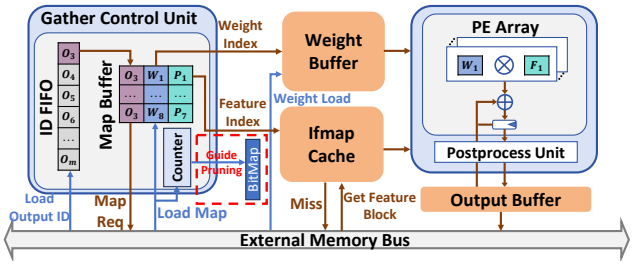


Figure 9: SpConv core. A bitmap is built using neighboring voxel counts to guide pruning in the attention core.

After retrieving map entries into the map buffer, the gather control unit loads corresponding weights and input features. These are then computed in the PE array, and the ifmap cache stores input features using consecutive addresses to avoid random access.

5.3 Attention Core

The attention core features a 64×16 PE array (as shown in Fig. 10 (b)) for the fine-grained dataflow in Sec. 4.3. Each $\mathbf{q}_i$ is sent to each row, and $\mathbf{K}/\mathbf{V}$ is sent to each column in blocks along the sequence dimension. When partitioning these blocks, $\mathbf{k}_j$ and $\mathbf{v}_j$ pruned by the position-aware pruning strategy in Sec. 4.2 are skipped according to the bitmap from the SpConv core. As shown in Fig. 10 (a), each $\mathbf{k}_j$ and $\mathbf{v}_j$ in the $\mathbf{K}/\mathbf{V}$ block is sequentially fed into the PE, and each PE in the array executes the four stages in Algo. 1. Moreover, stages 1-4 can be pipelined to maximize PE utilization. At the end of each row in the PE array, a summarizing unit combines the outputs of different PEs to obtain the final attention result for $\mathbf{q}_i$, as shown in Fig. 10 (c). Furthermore, the attention core reads a small portion of data from memory at a time and doesn't generate large intermediate results. This allows on-chip buffers to store only the currently calculated vectors instead of entire $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ and $\mathbf{S}$ matrices.

Inspired by Softermax [19], we use base-2 instead of base-e exponentiation for hardware-friendly computation. The exponential calculation is divided into integer and fractional parts. With s_{ij} always negative after subtracting the maximum value, integer parts are simplified to right shifts. Meanwhile, fractional parts are converted to MAC operations using a lookup table.

6 EVALUATION

6.1 Experimental Setup

We evaluated APTO's performance with two typical serialization-based models, Octformer [22] and PTv3 [23]. As shown in Tab. 1, evaluations were conducted on small-scale dataset ModelNet40 [24] for object classification, as well as large-scale datasets ScanNet [3] and S3DIS [1] for indoor scene semantic segmentation. To assess APTO's speedup and energy saving, we compared it against GPU and the state-of-the-art accelerators PointAcc [10] and SpOctA [13]. The computing resources of PointAcc, SpOctA and APTO were set to be equal for fairness. All models were implemented in PyTorch 1.12.1 and CUDA 11.8, and retrained to regain accuracy.

We implemented APTO's hardware design in Chisel, which was compiled to Verilog and validated through simulation. Synthesis was performed with Synopsys DC Compiler in TSMC 28*nm* process to generate area and power reports. CACTI [16] was used for simulating on-chip SRAM area and power consumption. Besides, a cycle-accurate simulator was developed to model APTO's hardware behavior and record read and write operations of on-chip SRAM, and integrated with DRAMSim3 [8] to simulate HBM2 behavior.

Table 1: Models and datasets.

Application	Dataset	Model	#Points
Classification	ModelNet40	Octformer- MN	1k
Segmentation	ScanNet	Octformer- SN PTv3- SN	100k
	S3DIS	PTv3- S3D	1M

Table 2: Impacts of the position-aware pruning strategy.

Model	Result	Loss	Sparsity	Window Size
Octformer- MN	91.75%	0.43%	32.20%	32
Octformer- SN	73.90%	0.52%	33.52%	32
PTv3- SN	76.63%	0.64%	34.02%	1024
PTv3- S3D	72.12%	0.30%	41.35%	128

6.2 Evaluation Results

6.2.1 Model Accuracy. To evaluate the effectiveness of the position-aware pruning strategy, we tested modified models with different attention window sizes, as detailed in Tab. 2. Voxels with more than 14 neighbors (half of all neighboring positions) are pruned. The results demonstrate that our pruning strategy allows models to achieve an average sparsity of 35.27% with less than 1% accuracy loss, exhibiting effectiveness on various datasets. Fig. 12 (a) shows the comparison with BigBird [26], which also doesn't need extra

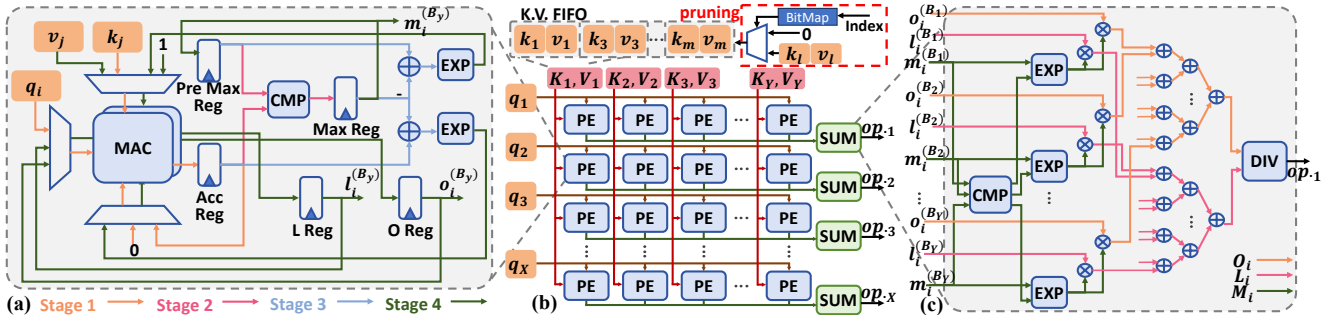


Figure 10: (a) Process element for fine-grained attention dataflow. (b) Attention PE array. (c) Summarizing unit.

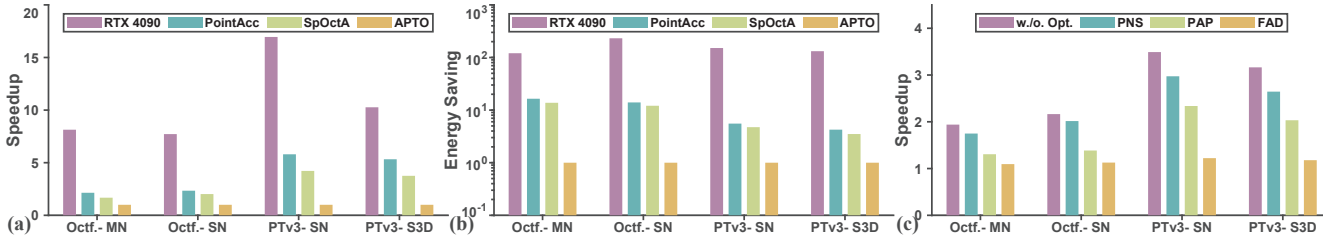


Figure 11: (a) Speedup results of APTO over RTX 4090 GPU, PointAcc, and SpOctA. (b) Energy saving results. (c) Ablation study results. (PNS: Parallel Neighbor Search, PAP: Position-Aware Pruning, FAD: Fine-Grained Attention Dataflow)

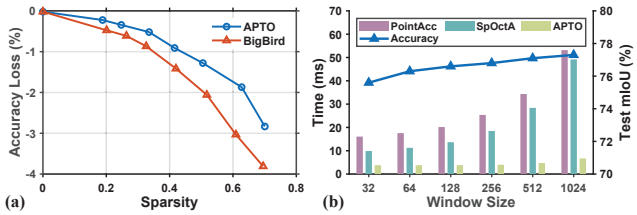


Figure 12: (a) Comparison of pruning strategies. (b) Accelerators' speedup on PTv3-SN as attention window size grows.

computations for pruning. Our method achieves higher accuracy under the same sparsity, making it more suitable for serialization-based point cloud transformers.

6.2.2 End-to-End Performance. Fig. 11 (a) and (b) demonstrate APTO's end-to-end speedup and energy saving results. Compared to RTX 4090 GPU, PointAcc, and SpOctA, APTO achieves average speedups of 10.22×, 3.53×, and 2.70×, along with energy savings of 153.59×, 8.57×, and 7.25×. APTO's performance enhancements are particularly significant in PTv3, achieving speedups of 5.80× and 4.22× for PTv3-SN, and 5.33× and 3.75× for PTv3-S3D, compared to PointAcc and SpOctA. In comparison, speedups for Octformer-MN are 2.14× and 1.67×, and for Octformer-SN are 2.34× and 2.02×. This is primarily because APTO's optimizations focus on attention computation, which constitutes a larger proportion of the computational cost in PTv3 than in Octformer, as shown in Fig. 3 (a).

6.2.3 Attention Optimization. Fig. 12 (b) illustrates APTO's capability for accelerating attention computation with different window sizes. As the window size increases, PointAcc and SpOctA encounter significant increases in execution time, while APTO shows only a slight increase. For a window size of 32, APTO achieves 4.24× and 2.60× speedups over PointAcc and SpOctA, and they increase to 7.95× and 7.35× when the window size reaches 1024. This capability of APTO for efficiently accelerating attention calculation in large-size windows is particularly advantageous for the state-of-the-art point cloud model [23], which seeks to improve accuracy by enlarging the attention window to broaden its receptive field.

6.2.4 Ablation Study. Fig. 11 (c) shows the breakdown of APTO's optimization effect. Implementing the Parallel Neighbor Search (PNS) for SpConv provides an average 1.14× speedup over the hardware design without optimizations in Fig. 4 by eliminating redundant attention computations. Additionally, Position-Aware Pruning (PAP) pushes the speedup to 1.52× by reducing data dependencies caused by softmax. Finally, the Fine-Grained Attention Dataflow (FAD) boosts the speedup to 2.26×.

Table 3: Overall comparisons with prior works.

	SpOctA	PointAcc	PTrAcc	Ours
Technology (nm)	40	40	65	28
Area (mm^2)	1.06	15.7	-	7.78
Frequency (MHz)	400	1024	1024	1024
On-chip SRAM (KB)	177.4	776	743	968
Peak Throughput (TOPS)	0.2	8	8	10
Bandwidth (GB/s)	16	256	256	256
Support SpConv	Yes	Yes	No	Yes
Support Transformer	No	No	Yes	Yes

6.2.5 Comparison with Other Works. Tab. 3 compares APTO's hardware configurations with PointAcc [10], SpOctA [13] and PTrAcc [9] (an accelerator for PT [27] based on FPS and kNN). APTO supports SpConv and transformer, leading to significant acceleration on serialization-based models compared to prior works. Its dedicated design and advanced process technology result in a smaller area footprint with higher throughput.

7 CONCLUSION

As point cloud deep learning gains popularity, the demand for dedicated hardware accelerators grows. We propose APTO, an accelerator for serialization-based point cloud transformers. It parallelizes neighbor searches in SpConv using z-curve indices, reduces redundant attention computations by pruning tokens of voxels with more neighbors, and employs fine-grained attention dataflow to minimize data dependencies. Experimental results show its effectiveness and efficiency on various serialization-based models.

ACKNOWLEDGMENTS

This work is supported in part by National Natural Science Foundation of China (NSFC) under grant No. (62002340, 62090020), Youth Innovation Promotion Association CAS under grant No. Y201923, and the Strategic Priority Research Program of CAS under grant No. XDB44030100. Part of this work is supported by the internship program of YUSUR Technology Co., Ltd. The corresponding author is Guihai Yan (yan@ict.ac.cn).

REFERENCES

[1] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 2016. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1534–1543.

[2] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 2019. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3075–3084.

[3] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5828–5839.

[4] Lei Dai, Shengwen Liang, Ying Wang, Huawei Li, and Xiaowei Li. 2024. APoX: Accelerate Graph-Based Deep Point Cloud Analysis via Adaptive Graph Construction. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 231–237.

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

[6] Meng Han, Liang Wang, Limin Xiao, Hao Zhang, Chenhao Zhang, Xilong Xie, Shuai Zheng, and Jin Dong. 2024. FuseFPS: Accelerating Farthest Point Sampling with Fusing KD-tree Construction for Point Clouds. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 238–243.

[7] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. 2022. Stratified transformer for 3d point cloud segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 8500–8509.

[8] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A cycle-accurate, thermal-capable DRAM simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109.

[9] Yaoxiu Lian, Xinhao Yang, Ke Hong, Yu Wang, Guohao Dai, and Ningyi Xu. 2023. A Point Transformer Accelerator with Fine-Grained Pipelines and Distribution-Aware Dynamic FPS. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[10] Yujun Lin, Zhekai Zhang, Haotian Tang, Hanrui Wang, and Song Han. 2021. Pointacc: Efficient point cloud accelerator. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 449–461.

[11] Zhijian Liu, Xinyu Yang, Haotian Tang, Shang Yang, and Song Han. 2023. Flatformer: Flattened window attention for efficient point cloud transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1200–1211.

[12] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. 2021. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 977–991.

[13] Dongxu Lyu, Zhenyu Lil, Yuzhou Chen, Jinming Zhang, Ningyi Xu, and Guanghui He. 2023. SpOctA: A 3D Sparse Convolution Accelerator with Octree-Encoding-Based Map Search and Inherent Sparsity-Aware Processing. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[14] Maxim Milakov and Natalia Gimelshein. 2018. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867* (2018).

[15] Guy M Morton. 1966. A computer oriented geodetic data base and a new technique in file sequencing. (1966).

[16] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.

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

[18] Zhuoran Song, Heng Lu, Gang Li, Li Jiang, Naifeng Jing, and Xiaoyao Liang. 2023. PRADA: Point Cloud Recognition Acceleration via Dynamic Approximation. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[19] Jacob R Stevens, Rangharajan Venkatesan, Steve Dai, Brucek Khailany, and Anand Raghunathan. 2021. Softermax: Hardware/software co-design of an efficient softmax for transformers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 469–474.

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

[21] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 97–110.

[22] Peng-Shuai Wang. 2023. Octformer: Octree-based transformers for 3d point clouds. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–11.

[23] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. 2024. Point Transformer V3: Simpler Faster Stronger. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4840–4851.

[24] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1912–1920.

[25] Xinhao Yang, Tianyu Fu, Guohao Dai, Shulin Zeng, Kai Zhong, Ke Hong, and Yu Wang. 2023. An efficient accelerator for point-based and voxel-based point cloud neural networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[26] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems* 33 (2020), 17283–17297.

[27] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. 2021. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*. 16259–16268.

[28] Jiapei Zheng, Hao Jiang, Xinkai Nie, Zhangcheng Huang, Chixiao Chen, and Qi Liu. 2023. TiPU: A Spatial-Locality-Aware Near-Memory Tile Processing Unit for 3D Point Cloud Neural Network. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[29] Yin Zhou and Oncel Tuzel. 2018. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4490–4499.

[30] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. 2021. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9939–9948.