FastPoint: Accelerating 3D Point Cloud Model Inference via Sample Point Distance Prediction

Donghyun Lee[1] Dawoon Jeong[1] Jae W. Lee[1] Hongil Yoon[1,2]
[1]Seoul National University [2]Google
{eudh1206, daun20211, jaewlee}@snu.ac.kr, hongilyoon@google.com

Abstract

Deep neural networks have revolutionized 3D point cloud processing, yet efficiently handling large and irregular point clouds remains challenging. To tackle this problem, we introduce FastPoint, a novel software-based acceleration technique that leverages the predictable distance trend between sampled points during farthest point sampling. By predicting the distance curve, we can efficiently identify subsequent sample points without exhaustively computing all pairwise distances. Our proposal substantially accelerates farthest point sampling and neighbor search operations while preserving sampling quality and model performance. By integrating FastPoint into state-of-the-art 3D point cloud models, we achieve 2.55× end-to-end speedup on NVIDIA RTX 3090 GPU without sacrificing accuracy.

1. Introduction

3D point clouds have become increasingly important for representing and understanding 3D scenes, driving advancements in fields like robotics and autonomous driving. Deep neural networks [4, 6, 7, 13, 17, 20, 23–26, 30, 32, 33, 37, 42, 43] have emerged as powerful tools for processing this type of data. While PointNet++ [24] and its successors have improved efficiency and performance of these point cloud models, the irregular nature and growing size of point cloud data continue to impose significant computational challenges, especially for real time use cases. Operations such as Farthest Point Sampling (FPS) and neighbor search remain major bottlenecks.

While various hardware accelerators [8, 9, 12, 16, 18, 36, 40] have been explored to accelerate the models, software-only solutions are less common. Given the high cost and complexity of developing specialized accelerators, software-based acceleration is a more practical approach. Recently, several software-based techniques [15, 39] have been proposed to accelerate these models through approximation. However, such approaches often come at the cost of significant accuracy loss, limiting their practical use.

To address this challenge, we propose *FastPoint, a novel software-based acceleration technique with sample point distance prediction*. We observe that the inherent nature of FPS, which iteratively selects the point farthest from the current set of sampled points, leads to two predictable trends in the minimum distances between these points:

1. *Decreasing minimum distance*: As FPS makes progress, the number of remaining points decreases, forcing the algorithm to select points closer to those already sampled. This results in a smoothly decreasing curve of minimum distances between sampled points.
2. *Early structure capture*: The initial sample points tend to be the extremities of the point cloud, effectively capturing its overall shape and boundaries.

Our proposal leverages these observations to accurately estimate the distance curve using only a few initial FPS iterations. By predicting the distance curve, we can efficiently identify subsequent sample points without computing and comparing pairwise distances at each FPS iteration. This leads to significant latency savings while maintaining sampling quality comparable to that of the original FPS. Furthermore, predicting the distance curve enables the sampling process to be decoupled from distance computation. This decoupling exposes new opportunities for parallelizing distance calculations in the proposed sampling, further enhancing efficiency. This approach not only accelerates FPS itself but also benefits subsequent operations such as neighbor search by leveraging pre-computed distances.

Our key contributions are summarized as follows:

- We identify two key latency bottlenecks in point cloud models: FPS and neighbor search. Limited parallelism in FPS and redundant distance computations across both operations are the primary sources of inefficiency.
- We empirically analyze 3D point cloud models and identify the decreasing trend and early structure capture trend.
- Based on the observations, we introduce a novel sampling approach with sample point distance prediction. With the estimated distance curve, this approach not only accelerates FPS itself but also benefits subsequent neighbor

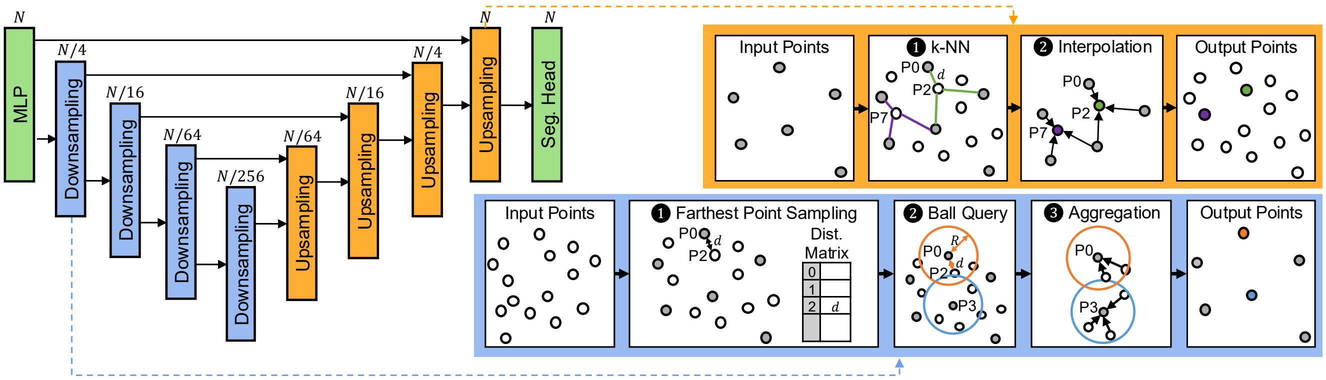


Figure 1. Overview of PointNet++ Based Model Architecture

search, which relies on pre-computed distances.

- We integrate FastPoint into OpenPoints library and experiment on state-of-the-art PointNet++ based models, achieving geomean 2.55× end-to-end speedup on NVIDIA RTX 3090 GPU while preserving accuracy.

2. Background and Related Work

2.1. Deep Learning on 3D Point Clouds

Deep Neural Networks on Raw Point Cloud Since the release of PointNet [23], which is the first to apply deep neural networks directly to the raw point clouds, numerous successors [6, 13, 17, 19, 24–27, 30, 31, 34, 41] have been developed, improving both model performance and efficiency. PointNet++ [24] introduces a hierarchical structure to PointNet through sampling and grouping, which has become the core architecture of subsequent PointNet++ based models. PointNeXt [27] revisits PointNet++, achieving substantial improvements in model performance, while PointVector [6] and PointMetaBase [17] push the model performance and efficiency even further. Although the architectural details of each model differ, all models follow the PointNet++ structure, utilizing Set Abstraction and Feature Propagation layers as core components. This work focuses on identifying and accelerating the primary latency bottlenecks in PointNet++ based models.

Deep Neural Networks on Voxelized Point Cloud Several works [3, 4, 10, 22, 28, 43] propose to voxelize point clouds, using voxels as input rather than raw points. The advent of 3D sparse convolution [4] has made this voxel-based approach increasingly popular due to its efficiency and high performance. Recently, transformer-based models [7, 21, 37, 38] operating on voxelized point clouds are gaining significant attention. Although voxelization alleviates the computational cost of mapping operations (i.e., downsampling and neighbor search), it has a notable limitation due to the loss of position information during the voxelization process.

2.2. Related Work

Numerous solutions have been proposed to accelerate sampling and neighbor search operations in PointNet++ based models. We categorize them into hardware- and purely software-based approaches.

Hardware-Assisted Acceleration QuickFPS [12] introduces a k-d tree based FPS algorithm which reduces computation and memory access in each FPS iteration, along with specialized hardware designed for this method. MARS [36] and PTrAcc [16] both propose hardware accelerators that employ a distance filtering technique which skips unnecessary distance computations in each iteration of FPS. Although these techniques achieve substantial speedups in FPS without any loss in sampling quality, they require specialized hardware to fully harness their algorithms, limiting their effectiveness on commodity hardware like GPU. For more thorough evaluation, we compare the speedup of our proposal with pure software version [11] of QuickFPS in Section 5.4, evaluating its effectiveness on GPU.

Software-Only Acceleration To mitigate the high latency associated with FPS, various alternative sampling methods have been explored. RandLA-Net [13] replaces FPS with random sampling and introduces a local feature aggregation module that compensates for the low sampling quality of random sampling. Grid sampling, the most commonly used alternative, is adopted by Grid-GCN [35] and KPConv [29], offering faster processing with reasonable sampling quality compared to random sampling. Despite the latency advantages of these methods, none have consistently outperformed FPS for overall performance. This is illustrated by the fact that most PointNet++-based models [6, 17, 24–27] continue to rely on FPS, highlighting its broad applicability and superior sampling quality. The experimental results in Appendix B.1 corroborate this claim.

EdgePC [39] structures the point cloud with morton codes to accelerate FPS and neighbor search on edge GPUs,

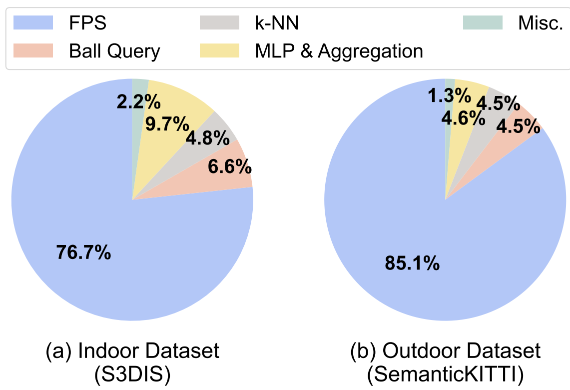


Figure 2. Latency Breakdown of PointNet++ Based Models

while Adjustable FPS [15] accelerates FPS by leveraging the intrinsic locality of the point cloud data. However, both methods demonstrate significant accuracy loss compared to the baseline (i.e., up to 2% mIoU loss) due to their aggressive approximations, limiting their effectiveness. Prior work [14] accelerates the training of PointNet++ based models by precomputing the minimum point spacing of farthest-sampled points before training and reusing this value across epochs for fast sampling. While this approach effectively speeds up training, it is not suitable for inference, as the minimum spacing value cannot be determined in advance in inference scenarios. Detailed comparisons with our work are provided in Appendix A.4.

3. Challenges of Point Cloud Models

3.1. Model Architecture Overview

Figure 1 illustrates the detailed process of PointNet++-based models. Similar to 2D convolutional neural networks, these models consist of both downsampling and upsampling layers. Commonly referred to as the set abstraction layer, the downsampling layer in PointNet++-based models comprises three main stages, as outlined below:

❶ **Farthest Point Sampling (FPS):** Downsampling begins by selecting $N/stride$ output points from N input points, where $stride$ represents the downsampling rate. Typically, the FPS is used to maintain the shape and structure of the input point cloud.

❷ **Ball Query:** Next, a ball query with radius R is performed around each sampled point to gather neighboring points for feature aggregation.

❸ **Aggregation:** The features of neighboring points identified in the ball query are then processed. A multi-layer perceptron (MLP) is applied to these features, followed by max pooling to aggregate the information for each centroid. The details of this process, e.g., positional embedding [17], can vary depending on the model architectures.

Algorithm 1 Farthest Point Sampling

Input P: input point cloud, n: number of points to sample, N: number of original points

Output sampled_idx: indices of sampled points

1: sampled_idx[0] $\leftarrow$ $seed\_idx$ // Starts with a random point
2: min_dists[i] $\leftarrow \infty$ for $i = 0, ..., N-1$
3: **for** $i \leftarrow 1$ to $n-1$ **do** // Not parallelized
4:     **for** $j \leftarrow 0$ to $N-1$ **do** // Parallelized
5:         $d_{new} \leftarrow \texttt{dist}(P[\text{sampled\_idx}[i-1]], P[j])$
6:         **if** min_dists[j] $> d_{new}$ **then**
7:             min_dists[j] $\leftarrow d_{new}$
8:     sampled_idx[i] $\leftarrow$ $\texttt{argmax}$(min_dists)

The upsampling layer, often referred to as a feature propagation layer, takes the downsampled points from the corresponding downsampling layer as inputs and propagates the features back to the original point cloud. This process is done by following steps.

❶ **k-NN:** For each upsampled point, k-NN is applied to identify its k nearest neighbors among the input points.

❷ **Interpolation:** Based on the k-NN results, neighboring input points are interpolated, concatenated with skip-connected features from the corresponding downsampling layer, and the information is passed through an MLP to propagate the features to the subsequent layer.

3.2. Latency Breakdown

Figure 2 presents the latency breakdown of PointMetaBase-L model evaluated on both indoor (S3DIS) and outdoor (SemanticKITTI) scenes. The results show that coordinate-based operations (i.e., FPS, k-NN, and Ball Query) dominate the execution time, consuming significantly more time than feature-based operations (i.e., Aggregation and MLP). These coordinate-based operations account for approximately 88.1% of the total inference time for indoor scenes and 94.1% for the outdoor scenes. Among these, FPS emerges as the primary performance bottleneck, requiring the most execution time. Additionally, neighbor search operations, including k-NN and Ball Query, contribute significantly to the overall latency. To address these challenges, we first analyze the root causes of these inefficiencies and propose novel optimization techniques in subsequent sections.

3.3. Inefficiencies in Farthest Point Sampling

Farthest point sampling operates iteratively by selecting one point at a time. In each iteration, the sampling algorithm identifies the point farthest from the set of the previously sampled points. The detailed process is outlined in Algorithm 1. Initially, a random point in P is chosen as the seed point (Line 1) and added to the *sampled_idx*. Then, a distance matrix (i.e., *min_dists*) is created to store the distance between each point in P and the set of the previously sampled points, where the distance from a point to a point set is defined as minimum distance between the point and any point in the set. This matrix is initialized to ∞ (Line 2). In each iteration, the distance between the most recently sampled point and every other point in P is calculated (Lines 3-7). If a newly calculated distance is smaller than the corresponding entry in the *min_dists* matrix, the matrix is updated with the smaller value. Finally, the index of the point with the maximum distance in the *min_dists* is added to the *sampled_idx* (Line 8). This process is repeated until the specified number of points (n) is sampled.

The sequential nature of the FPS algorithm is the primary reason of the inefficiency. Because only one point is sampled per iteration, the $O(N)$ distance calculations required in each iteration must be sequentially executed n times. This tight coupling between sampling and distance calculation significantly limits the potential for parallelism in distance computations.

To unlock this parallelism potential, we introduce *Minimum Distance Prediction Sampling (MDPS)* in Section 4.1, a novel sampling strategy that fully decouples sampling from distance computation, enabling greater parallelism and faster processing speeds.

3.4. Inefficiencies in Neighbor Search

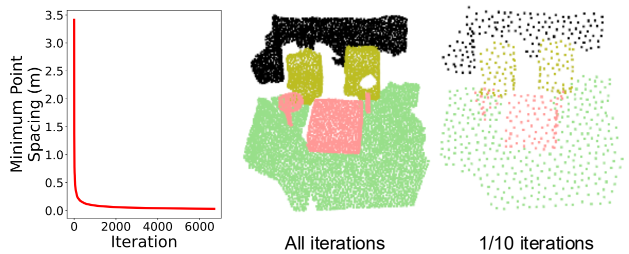
As explained in Section 3.1, PointNet++ based models employ two types of neighbor search algorithms: Ball Query and k-NN. These operations play a critical role in capturing local geometric relationships in the point cloud. Ball Query identifies neighbor points within a fixed radius around the downsampled points. k-NN identifies k nearest neighbors among the downsampled points for each original point.

The inefficiencies in these neighbor search operations stem from redundant distance computations in both the FPS and subsequent neighbor search operations. For instance, as shown in Figure 1, the distance between P0 and P2 is computed while updating P2's minimum distance in FPS. However, the same distance is redundantly recalculated in both Ball Query and k-NN. Section 4.2 introduces *redundancy-free neighbor search* techniques to address this issue.

4. FastPoint: Fast Point Cloud Inference

Trends in Minimum Distance of Sample Points The FPS iteratively selects the point farthest from the current set of sampled points, which leads to predictable trends in the distance between the sampled points.

1. *Decreasing minimum distance*: As sampling proceeds, the pool of the remaining points shrinks, the FPS algorithm selects points closer to those already sampled over time. Thus, the maximized minimum distance (i.e., `max`(*min_dists*) in Algorithm 1) decreases, which we define as *minimum distance* for brevity. This is particularly evident in Figure 3a, showing a smooth, decreasing curve for various input point clouds.
2. *Early structure capture*: Importantly, we observe that a few initial sample points are sufficient to capture the overall shape and boundaries of the input point cloud (Figure 3b). This is because the initial sample points tend to be the extremities of the point cloud.



(a) Minimum Distance Curve of ScanNet scene (b) Visualization of ScanNet scene sampled at 1/10 of the original iterations

Figure 3. Motivation of Minimum Distance Curve Estimation

Optimization Opportunities The trend of *early structure capture* suggests that the initial iterations of FPS are important in accurately representing the point cloud's structure. The *Decreasing trend* suggests that the later portion of the minimum distance curve becomes more predictable as sampling progresses. Building upon these trends, if we could accurately estimate this curve using only a few initial FPS iterations, we could maintain a comparable sampling quality while achieving substantial latency reduction by selecting points guided by the predicted curve.

Proposed Techniques We introduce two optimizations by leveraging these opportunities: Minimum Distance Prediction Sampling and Redundancy-Free Neighbor Search. Minimum Distance Prediction Sampling (MDPS) is a novel sampling strategy designed to approximate the sampling quality of FPS while significantly reducing latency. This approach estimates the distance curve using only a few initial non-parallelizable FPS iterations. Subsequent sample points are selected based on the predicted distances, eliminating the need for pairwise distance comparisons in later iterations. Furthermore, MDPS enables Redundancy-Free Neighbor Search by reusing precomputed distance information, thereby eliminating redundant computations during subsequent Ball Query and k-NN operations. This optimization further enhances the overall efficiency of PointNet++-based models during inference.

4.1. Minimum Distance Prediction Sampling

❶ **Minimum Distance Curve Estimation** To build a low-cost estimator for the minimum distance curve (as shown in Figure 4), we aim to estimate the curve with minimal error using the results from as few FPS iterations as possible. We

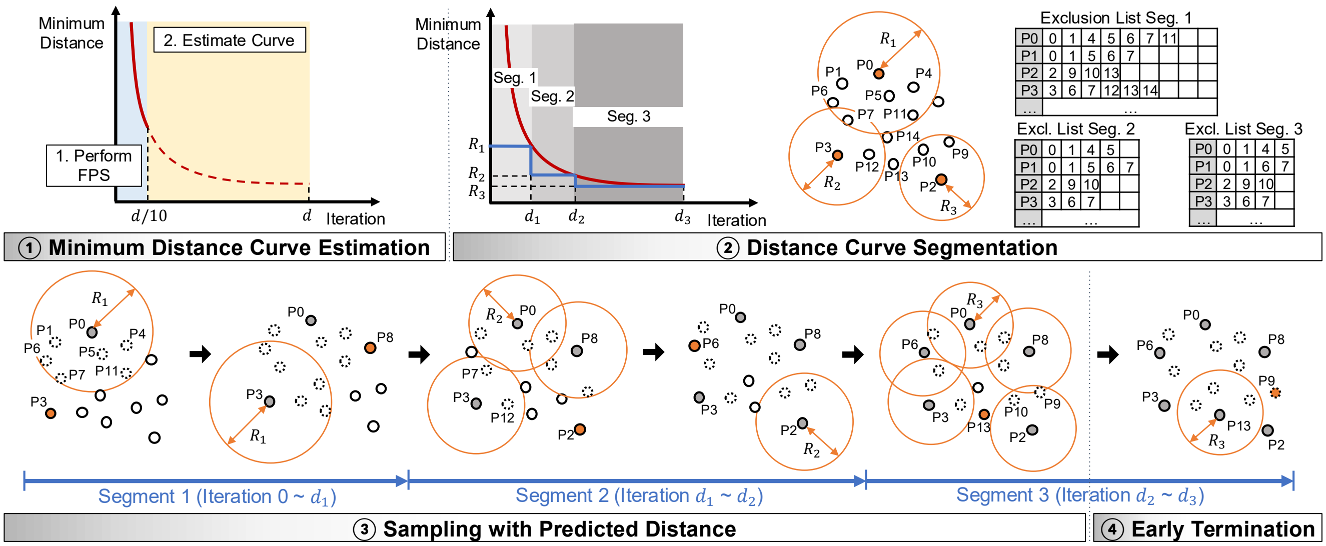


Figure 4. Overall Flow of Minimum Distance Prediction Sampling

formulate the estimator $\mathcal{E}$ as a function which gets initial p segment (where p is the ratio of initial FPS iterations to total iterations, $0 < p < 1$) of the curve C as input and outputs the rest of the estimated curve $\hat{C}$ (i.e., Equation 1). Note that $C(1)$ represents the minimum distance among the sampled points after all sampling iterations are finished.

$$\hat{C}(t) = \mathcal{E}\left(C(t) \mid t \in [0, p]\right) \quad \text{for } t \in (p, 1] \tag{1}$$

We have experimented with various models, including polynomial functions and MLPs, and find that MLPs yield the best performance (Appendix A.3). We train a 3-layer MLP on minimum distance curves extracted from the training split of each point cloud dataset, setting $p = 0.1$ to balance error and efficiency. Our estimator achieves a low Mean Absolute Percentage Error (MAPE) of 1.02%, 0.77%, and 1.93% on S3DIS, ScanNet, and SemanticKITTI datasets, respectively, performing better on indoor datasets (S3DIS, ScanNet) due to their more consistent point density and distribution. Appendix B.6 and B.8 discuss ablation study for p and cross-dataset applicability of estimators.

❷ Distance Curve Segmentation Ideally, what we want is to sample points whose minimum distance matches the value on the curve at each iteration. However, directly finding the point that yields the exact minimum distance in every iteration is computationally expensive.

To efficiently sample points along the estimated minimum distance curve, we first divide the curve into multiple segments. Note that each segment is associated with a group of consecutive iterations. Then, the corresponding predicted minimum distances are employed to identify sample point at the segment-level granularity in the sampling process.

As illustrated in Figure 4, the curve is segmented at iterations d_1, d_2, and d_3, resulting in three segments. For each input point, we identify points within a specified radius of each segment boundary (i.e, R_1, R_2, and R_3) and store them in the exclusion list for three segments. This list prevents sampling closely located points by considering the sampled points.

To optimize computational efficiency, building the exclusion list across all segments is fused into a single GPU kernel. By calculating the distance between the points only once, regardless of the number of segments, we minimize the latency overhead. For example, the distance between P0 and P1 is computed only once. If P1 is within R_3 (i.e., dist(P0, P1) $< R_3 < R_2 < R_1$), P1 is added to the exclusion list of P0 at all segments. Refer to Table 3 for latency overhead with an increase in the segment count.

Building the exclusion list still requires computing pairwise distances; however, this process is much faster than the original FPS algorithm. This improvement is achieved because all computations are fully parallelizable, as sampling and distance calculations are completely decoupled. For details on the algorithm, refer to Appendix A.1.

❸ Sampling with Predicted Distance The sampling process employs the exclusion list. Starting from Segment 1, a seed point (P0 in Figure 4) is chosen. In each sampling iteration, points within a radius R_1 from the most recently sampled point are filtered out based on the exclusion list. We employ bitmaps to record which points are available for sampling. For instance, in the first iteration of Figure 4, Point P0, P1, P4, P5, P6, P7, and P11 are excluded based on the exclusion list of Segment 1. Point P3 is randomly selected among the eight available points for the second iteration in the same segment. Excluded points remain unavailable until the current segment is complete.

As the sampling progresses and the segment changes

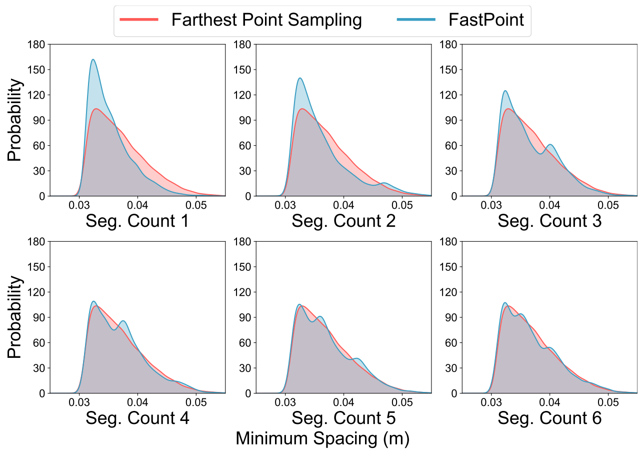


Figure 5. Minimum spacing distribution comparison with increasing segment counts.

(e.g., from Segment 1 to Segment 2), the radius threshold is adjusted (e.g., from R_1 to R_2). This transition changes the set of points that are available for sampling. By iteratively applying this process, we ensure that the distance between sampled points meets or exceeds the threshold for each segment. This results in a minimum distance that lies above the blue step function in the segmented distance curve (i.e., R_i serves as the lower bound of the minimum distance in Segment i). Increasing the number of segments allows the step function to approach the red minimum distance curve, improving sampling quality and converging towards the quality of FPS. Figure 5 demonstrates this convergence. With more segments, the distribution of minimum distance between sampled points becomes closer to that of FPS. Details on implementation are provided in Appendix A.1.

❹ **Early Termination** While the estimated minimum distance curve exhibits low error, overestimation can still negatively impact sampling quality. This is because it can lead to an excessive number of point exclusions, limiting the availability of points in later iterations. To mitigate this issue, we introduce Early Termination. This technique monitors the availability of points at each iteration. If no points remain, it advances to the next segment earlier than planned. When the final segment is exhausted, Sampling with Predicted Distance is terminated, and the remaining iterations are completed using the original FPS algorithm. For seamless transition to FPS, we efficiently update the minimum distance matrix ($dists$ in Algorithm 1) by leveraging the exclusion list. This approach avoids recalculating all-to-all distances between points, significantly reducing the search space and minimizing the impact on sampling time. Appendix A.1 describes the details of Early Termination.

Factors Contributing to Speedup The primary factors contributing to the latency of MDPS are:

1. *Minimum Distance Curve Estimation:* This step, requiring 1/10 of the original FPS iterations, adds only a fraction of the original FPS latency.
2. *Exclusion List Construction:* The required all-to-all distance calculations are fully parallelizable, resulting in minimal latency overhead. Detailed analysis on this factor is provided in Appendix A.2.
3. *Sampling Process:* The use of exclusion list eliminates the need for additional distance calculations. Early termination requires additional FPS iterations. However, the impact is minimal: the overhead of 2.02%, 0.58%, and 1.18% of original FPS iterations in S3DIS, ScanNet, and SemanticKITTI dataset, respectively.

Due to these optimizations, MDPS can achieve a 4-5× speedup relative to the original FPS algorithm. Detailed latency breakdown of MDPS is provided in Appendix B.2.

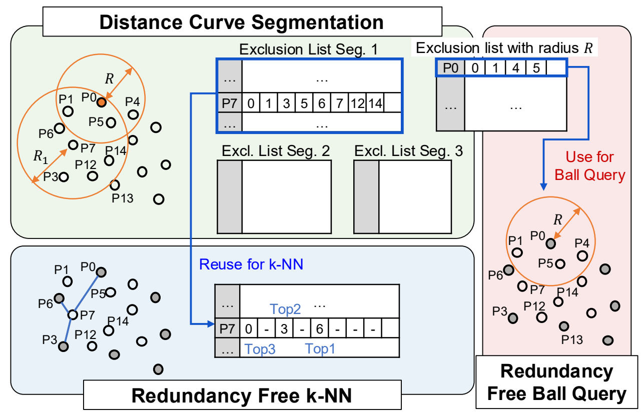


Figure 6. Redundancy Free Neighbor Search

4.2. Redundancy-Free Neighbor Search

The exclusion list in MDPS captures spatial relationships between points, providing an opportunity to optimize subsequent Ball Query and k-NN operations. We introduce Redundancy Free Ball Query and k-NN, which fully leverage the exclusion list from the sampling stage.

Redundancy-Free Ball Query Ball query, which identifies neighbors within a radius R, is performed after sampling, before aggregation (refer to Figure 1). The exclusion list is well-suited for this task, as it contains points within a specified radius of each point. By adding an extra list to the exclusion list for the radius R, as illustrated in Figure 6, we can directly use it for ball query. As discussed in Section 4.1, this approach minimizes computational overhead; distances between points are calculated only once, regardless of the number of segments. After sampling, we extract the relevant entries from the corresponding exclusion list for the ball query, which serve as centroids for aggregation.

Redundancy-Free k-NN While directly reusing the exclusion list for k-NN is challenging, we can leverage it to reduce the search space. By using the exclusion list of Seg-

Dataset	Average Minimum Distance (Ratio to Baseline)			
	Baseline (FPS)	Random Sampling	Grid Sampling	MDPS (Ours)
S3DIS	0.06395	0.03478 (54.38%)	0.04971 (77.72%)	0.06359 (99.45%)
ScanNet	0.03701	0.02035 (54.99%)	0.02400 (64.84%)	0.03677 (99.35%)
SemanticKITTI	0.25505	0.12775 (50.09%)	0.21519 (84.37%)	0.25087 (98.36%)

Table 1. Comparison of Sampling Quality among FPS, Random Sampling, Grid Sampling and MDPS.

ment 1, we can limit the search space by eliminating points that are too far away to be potential k-NN neighbors. For example, to find the three nearest neighbors of point P7 in Figure 6, we calculate distances between P7 and the points from the corresponding exclusion list to identify the three nearest neighbors. Because k-NN upsampling relies on the downsampled set, points that were not downsampled are excluded from the search space. This approach significantly reduces the number of distance calculations, leading to substantial latency reduction.

5. Evaluation

5.1. Methodology

We evaluate FastPoint on PointMetaBase [17] and PointVector [6], two recent models based on PointNet++. We evaluate our proposal on both indoor and outdoor datasets to demonstrate its scalability. We use S3DIS [1] and ScanNet [5] for indoor datasets and use SemanticKITTI [2] for outdoor datasets.

The end-to-end latency is measured for processing all scenes in each dataset's validation split. We use mean Intersection over Union (mIoU) as a model accuracy evaluation metric. Detailed evaluation methodologies follow the same approach in PointMetaBase [17], and PointVector [6]. We apply FastPoint and the other baseline methods only to the first layer of each model as the first layer's FPS and neighbor search dominates the total computation time (>90%).

We implement FastPoint with custom CUDA kernels and integrate them into OpenPoints library. We also integrate the software implementation [11] of QuickFPS [12] open-sourced by the authors to OpenPoints for comparison. The experiments are conducted using an NVIDIA Geforce RTX 3090 GPU with 24GB of memory. We use CUDA 11.8 and PyTorch 1.10.1 for software setup.

5.2. Sampling Quality of MDPS

To evaluate sampling quality, we compute the average minimum distance between neighboring sampled points. A larger average distance indicates better coverage of sample points from the perspectives of point distribution and structure preservation. As shown in Table 1, MDPS achieves sampling quality comparable to FPS, with an average minimum distance reaching over 99% of the FPS baseline for S3DIS and ScanNet, and over 98% for SemanticKITTI. In contrast, Random Sampling and Grid Sampling [29] exhibit significantly lower quality, with their average minimum distances deviating substantially from the FPS baseline. We also compare the minimum spacing distribution of each sampling method via visualization in Appendix B.9 and evaluate the robustness of MDPS in Appendix B.7.

Model	Dataset	mIoU (Diff. to Baseline FPS)			
		Baseline (FPS)	Random Sampling	Grid Sampling	MDPS (Ours)
PV-L	S3DIS	71.33	64.83 (-6.5)	70.97 (-0.36)	71.37 (+0.04)
	ScanNet	70.70	59.94 (-10.76)	69.61 (-1.09)	70.63 (-0.07)
	Semantic KITTI	50.91	38.46 (-12.45)	50.62 (-0.29)	50.79 (-0.12)
PMB-L	S3DIS	69.72	68.21 (-1.51)	69.54 (-0.18)	69.74 (+0.02)
	ScanNet	70.86	67.66 (-3.2)	70.36 (-0.50)	70.89 (+0.03)
	Semantic KITTI	52.19	47.26 (-4.93)	52.27 (+0.08)	52.09 (-0.10)

Table 2. Accuracy comparison. PV and PMB stand for PointVector and PointMetaBase.

5.3. Accuracy Impact of MDPS

We analyze the impact of MDPS on accuracy by applying it to the inference stage of PointNet++ models initially trained with FPS. Since MDPS is designed to closely replicate the FPS sampling pattern, it is compatible with FPS-trained models, thereby minimizing accuracy degradation. Note that Redundancy-Free Neighbor Search does not introduce any approximation, thus has no impact on accuracy. To compare the effects of different sampling methods, we also evaluate random sampling and grid sampling[1].

Table 2 shows that MDPS excels in maintaining accuracy, with negligible differences observed across all datasets and models. This highlights its effectiveness in replicating FPS behavior. However, the grid sampling exhibits larger and less consistent accuracy drops. When applied only during inference, it leads to declines of up to 1.09% (PointVector-L, ScanNet). While it outperforms FastPoint in a single instance (PointMetaBase-L, SemanticKITTI), its performance is generally inferior, with significant accuracy reductions in other scenarios. The random sampling shows considerable accuracy drops across all data points.

Overall, FastPoint effectively preserves accuracy by closely approximating the sampling pattern of FPS, achieving accuracy levels near-identical to FPS without retraining. Latency-accuracy comparisons (Appendix B.5) show that FastPoint resides above the pareto front, emphasizing the effectiveness of FastPoint.

[1]Grid sampling is applied only during inference. Appendix B.1 shows results for cases where both training and inference utilize grid sampling.

Method	Number of Segments	Evaluation Metrics		
		Accuracy (mIoU)	Sampling Time (ms)	Sampling Quality
FPS	-	70.70	85.83	0.03701
MDPS (Ours)	1 Segment	70.34	18.62	0.03532
	2 Segments	70.60	19.43	0.03598
	3 Segments	70.46	19.79	0.03636
	4 Segments	70.54	21.01	0.03662
	5 Segments	70.52	21.41	0.03665
	6 Segments	70.63	22.06	0.03662
	7 Segments	70.63	22.75	0.03664

Table 3. Comparison of mIoU, sampling time, and sampling quality for FPS and MDPS across segments 1 to 7. Sampling time is measured with a single scene of ScanNet dataset. PointVector-L model is used for mIoU comparison.

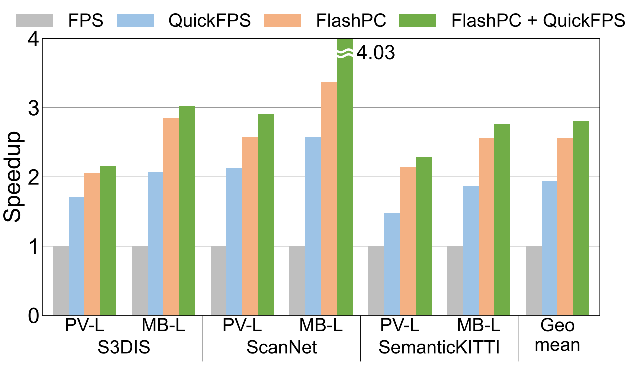


Figure 7. End-to-end speedup of FastPoint and QuickFPS.

5.4. Latency Reduction of FastPoint

To evaluate the impact of FastPoint on end-to-end latency, we compare its latency to baseline FPS and QuickFPS. Note that FastPoint includes both MDPS and Redundancy-Free Neighbor Search. As explained in Section 2.2, QuickFPS accelerates FPS using k-d tree without approximation. Since MDPS also contains FPS steps when predicting the minimum distance curve, we can achieve further latency reductions by replacing them with QuickFPS, meaning that FastPoint and QuickFPS can be synergetic.

Figure 7 presents the results of this comparison. Across S3DIS, ScanNet, and SemanticKITTI datasets, FastPoint achieves substantial latency reductions, with a geomean end-to-end speedup of 2.55× compared to baseline FPS. Integrating QuickFPS with FastPoint further reduces the execution latency achieving a geomean end-to-end speedup of 2.76× compared to baseline FPS and 1.41× compared to standalone QuickFPS. Speedup specific to the sampling and neighbor search are provided in Appendix B.3 and B.4.

These findings substantiate the effectiveness of FastPoint in reducing latency, surpassing existing methods like QuickFPS. Moreover, the successful combination of FastPoint with QuickFPS demonstrates its compatibility with existing FPS optimizations. Scalability to Non-PointNet++ based models are discussed in Appendix B.10.

Model	Dataset	Method	Speedup
PV-L	S3DIS	MDPS	1.90×
		All	1.98×
	ScanNet	MDPS	2.33×
		All	2.52×
	Sematic KITTI	MDPS	2.07×
		All	2.20×
PMB-L	S3DIS	MDPS	2.50×
		All	2.82×
	ScanNet	MDPS	2.89×
		All	3.38×
	Sematic KITTI	MDPS	2.31×
		All	2.62×

(a) Comparison of component-wise end-to-end speedup of FastPoint.

Model	# of Points	Speedup
PV-L	16,000	1.33×
	32,000	2.41×
	48,000	2.68×
	64,000	3.01×
	80,000	3.25×
	96,000	3.40×
PMB-L	16,000	1.58×
	32,000	2.64×
	48,000	2.91×
	64,000	3.06×
	80,000	3.30×
	96,000	3.44×

(b) Comparison of speedup as the number of points increases.

Table 4. Ablation study for speedup breakdown and scalability.

5.5. Ablation Study

Segment Count We evaluate the impact of varying the segment count on accuracy, sampling time, and sampling quality (i.e, average minimum distance). Table 3 shows that while sampling time increases gradually with more segments due to the exclusion list construction (refer to Section 4.1), both sampling quality and accuracy improve. However, they get saturated beyond a certain number of segments. Based on the empirical analysis, we use 6 segments by default, where accuracy saturates for all workloads.

Component-wise Speedup To understand the individual contributions of each component in FastPoint, we evaluate the end-to-end speedup by sequentially applying MDPS and Redundancy Free Neighbor Search. Table 4a shows that MDPS alone provides significant speedup. Incorporating Redundancy Free Neighbor Search further provides additional improvement, demonstrating that each component makes a distinct contribution to the overall speedup.

5.6. Scalability Study

To assess the scalability of FastPoint, we evaluate the end-to-end speedup compared to FPS by varying point cloud sizes. Table 4b shows that FastPoint consistently achieves higher speedups as the number of points increases. This trend highlights the efficiency of our approach, particularly when processing larger point clouds where the computational cost of the FPS becomes a significant bottleneck.

6. Conclusion

3D point clouds are critical for representing and understanding 3D scenes, and deep learning models like PointNet++ have shown great promise in processing this data. However, computational efficiency remains a key challenge. We propose FastPoint, a software acceleration technique that leverages predictable minimum distances between sampled points to efficiently identify subsequent points without exhaustive computations. Our evaluation shows a 2.55× speedup over baseline FPS while maintaining accuracy, enabling low-cost deployment of PointNet++ based models.

Acknowledgements

This work was supported by the Ministry of Science and ICT (MSIT), Korea, under the following grants: the Global Scholars Invitation Program (RS-2024-00456287), the Graduate School of Artificial Intelligence Semiconductor grant (IITP-2025-RS-2023-00256081), and the Artificial Intelligence Innovation Hub (No. 2021-0-02068), all supervised by the Institute for Information & Communications Technology Planning & Evaluation (IITP). Additional support was provided by the National Research Foundation of Korea (NRF) grant (RS-2024-00405857) funded by the Korea government (MSIT), and the Google Faculty Research Award. The source code is available at https://github.com/SNU-ARC/FastPoint.git. Jae W. Lee and Hongil Yoon are the corresponding authors.

References

[1] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *CVPR*, 2016. 7

[2] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Juergen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *ICCV*, 2019. 7

[3] Yukang Chen, Jianhui Liu, Xiangyu Zhang, Xiaojuan Qi, and Jiaya Jia. Largekernel3d: Scaling up kernels in 3d sparse cnns. In *CVPR*, 2023. 2

[4] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019. 1, 2

[5] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 7

[6] Xin Deng, WenYu Zhang, Qing Ding, and XinMing Zhang. Pointvector: A vector representation in point cloud analysis. In *CVPR*, 2023. 1, 2, 7

[7] 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

[8] Yu Feng, Boyuan Tian, Tiancheng Xu, Paul Whatmough, and Yuhao Zhu. Mesorasi: Architecture support for point cloud analytics via delayed-aggregation. In *Proceedings of the 53th International Symposium on Microarchitecture (MICRO)*, 2020. 1

[9] Yu Feng, Gunnar Hammonds, Yiming Gan, and Yuhao Zhu. Crescent: Taming memory irregularities for accelerating deep point cloud analytics. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022. 1

[10] Lei Han, Tian Zheng, Lan Xu, and Lu Fang. Occuseg: Occupancy-aware 3d instance segmentation. In *CVPR*, 2020. 2

[11] Meng Han, Liang Wang, Limin Xiao, Hao Zhang, Chenhao Zhang, Xiangrong Xu, and Jianfeng Zhu. Quickfps. http://github.com/hanm2019/bucket-based_farthest-point-sampling_GPU. 2, 7

[12] Meng Han, Liang Wang, Limin Xiao, Hao Zhang, Chenhao Zhang, Xiangrong Xu, and Jianfeng Zhu. Quickfps: Architecture and algorithm co-design for farthest point sampling in large-scale point clouds. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023. 1, 2, 7

[13] 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. 1, 2

[14] Donghyun Lee, Yejin Lee, Jae W. Lee, and Hongil Yoon. Frugal 3d point cloud model training via progressive near point filtering and fused aggregation. In *ECCV*, 2024. 3

[15] Jingtao Li, Jian Zhou, Yan Xiong, Xing Chen, and Chaitali Chakrabarti. An adjustable farthest point sampling method for approximately-sorted point cloud data. In *2022 IEEE Workshop on Signal Processing Systems (SiPS)*, 2022. 1, 3

[16] Yaoxiu Lian, Xinhao Yang, Ke Hong, Yu Wang, Guohao Dai, and Ningyi Xu. A point transformer accelerator with fine-grained pipelines and distribution-aware dynamic fps. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023. 1, 2

[17] Haojia Lin, Xiawu Zheng, Lijiang Li, Fei Chao, Shanshan Wang, Yan Wang, Yonghong Tian, and Rongrong Ji. Meta architecure for point cloud analysis. In *CVPR*, 2023. 1, 2, 3, 7

[18] Xueyuan Liu, Zhuoran Song, Guohao Dai, Gang Li, Can Xiao, Yan Xiang, Dehui Kong, Ke Xu, and Xiaoyao Liang. Fusionarch: A fusion-based accelerator for point-based point cloud neural networks. In *ACM/IEEE Design, Automation and Test in Europe Conference (DATE)*, 2024. 1

[19] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *ICCV*, 2019. 2

[20] Zhijian Liu, Xinyu Yang, Haotian Tang, Shang Yang, and Song Han. Flatformer: Flattened window attention for efficient point cloud transformer. In *CVPR*, 2023. 1

[21] 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. 2

[22] Alexey Nekrasov, Jonas Schult, Or Litany, Bastian Leibe, and Francis Engelmann. Mix3D: Out-of-Context Data Augmentation for 3D Scenes. In *International Conference on 3D Vision (3DV)*, 2021. 2

[23] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016. 1, 2

[24] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 1, 2

[25] 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.

[26] Guocheng Qian, Hasan Hammoud, Guohao Li, Ali Thabet, and Bernard Ghanem. Assanet: An anisotropical separable set abstraction for efficient point cloud representation learning. In *NeurIPS*, 2021. 1

[27] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. Pointnext: Revisiting pointnet++ with improved training and scaling strategies. In *NeurIPS*, 2022. 2

[28] Danila Rukhovich, Anna Vorontsova, and Anton Konushin. Imvoxelnet: Image to voxels projection for monocular and multi-view general-purpose 3d object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022. 2

[29] 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, 7

[30] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019. 1, 2

[31] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, 2019. 2

[32] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. In *NeurIPS*, 2022. 1

[33] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point transformer v3: Simpler, faster, stronger. In *CVPR*, 2024. 1

[34] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *CVPR*, 2021. 2

[35] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. Grid-gcn for fast and scalable point cloud learning, 2020. 2

[36] Xinhao Yang, Tianyu Fu, Guohao Dai, Shulin Zeng, Kai Zhong, Ke Hong, and Yu Wang. An efficient accelerator for point-based and voxel-based point cloud neural networks. In *ACM/IEEE Design Automation Conference (DAC)*, 2023. 1, 2

[37] Yu-Qi Yang, Yu-Xiao Guo, Jian-Yu Xiong, Yang Liu, Hao Pan, Peng-Shuai Wang, Xin Tong, and Baining Guo. Swin3d: A pretrained transformer backbone for 3d indoor scene understanding. *arXiv preprint arXiv:2304.06906*, 2023. 1, 2

[38] Yu-Qi Yang, Yu-Xiao Guo, and Yang Liu. Swin3d++: Effective multi-source pretraining for 3d indoor scene understanding. *arXiv preprint arXiv:2304.06906*, 2024. 2

[39] Ziyu Ying, Sandeepa Bhuyan, Yan Kang, Yingtian Zhang, Mahmut T. Kandemir, and Chita R. Das. Edgepc: Efficient deep learning analytics for point clouds on edge devices. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023. 1, 2

[40] Hyunsung Yoon and Jae-Joon Kim. Efficient sampling and grouping acceleration for point cloud deep learning via single coordinate comparison. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023. 1

[41] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. PointWeb: Enhancing local neighborhood features for point cloud processing. In *CVPR*, 2019. 2

[42] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer. In *ICCV*, 2021. 1

[43] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. *arXiv preprint arXiv:2011.10033*, 2020. 1, 2