Flash3D: Super-scaling Point Transformers through Joint Hardware-Geometry Locality

Liyan Chen[1], Gregory P. Meyer[2], Zaiwei Zhang[2], Eric M. Wolff[2], Paul Vernaza[2]
[1]The University of Texas at Austin
[2]Cruise LLC
liyanc@cs.utexas.edu, eric.wolff@getcruise.com, paul.vernaza@getcruise.com
https://github.com/liyanc/Flash3DTransformer

Abstract

Recent efforts recognize the power of scale in 3D learning (e.g. PTv3) and attention mechanisms (e.g. FlashAttention). However, current point cloud backbones fail to holistically unify geometric locality, attention mechanisms, and GPU architectures in one view. In this paper, we introduce ***Flash3D Transformer****, which aligns geometric locality and GPU tiling through a principled locality mechanism based on Perfect Spatial Hashing (PSH). The common alignment with GPU tiling naturally fuses our PSH locality mechanism with FlashAttention at negligible extra cost. This mechanism affords flexible design choices throughout the backbone that result in superior downstream task results. Flash3D outperforms state-of-the-art PTv3 results on benchmark datasets, delivering a 2.25x speed increase and 2.4x memory efficiency boost. This efficiency enables scaling to wider attention scopes and larger models without additional overhead. Such scaling allows Flash3D to achieve even higher task accuracies than PTv3 under the same compute budget.*

1. Introduction

Efficient and scalable processing of point clouds is crucial for a wide range of applications, including autonomous driving [14, 15, 27, 35, 44], robotic navigation [24, 29, 38, 42, 45], and augmented reality [11, 30]. Point cloud backbones are essential architectures that extract meaningful features from raw 3D data. Recent advancements [17, 18, 22, 39, 41] have explored various strategies to enhance the performance of these backbones, particularly focusing on how Multi-Head Self-Attention (MHSA) mechanisms can be optimized for 3D point clouds. Windowing MHSA and region shifting (e.g., Swin Transformers [21, 43]) enable efficient local-global feature extraction, but scaling point cloud backbones to larger point cloud sizes and model capacities remains challenging [21, 39, 41].

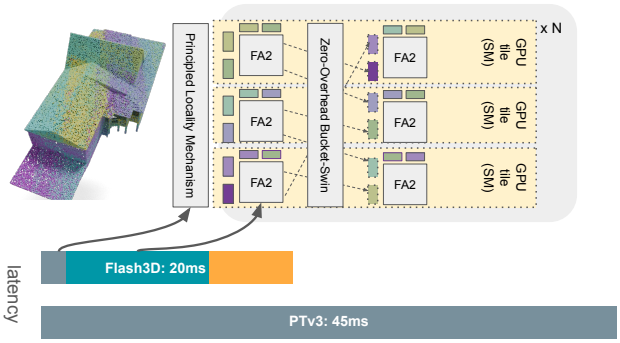


Figure 1. Effectiveness our Flash3D transformer by unifying geometric locality, FlashAttention (FA2), and GPU tiling architecture. Our unified perspective leads to drastically improved speed and scalability of point transformers.

In this paper, we tackle the challenges of efficient and scalable point cloud backbones by introducing **Flash3D Transformer**, a new architecture that unifies geometric locality with GPU memory locality. This is achieved through a principled locality mechanism based on Perfect Spatial Hashing (PSH). Flash3D is designed to align point cloud backbones with the tiling architecture of GPUs, leading to marked improvements in both efficiency and scalability. Our work integrates a locality mechanism that bridges geometric and GPU memory locality, enabling an efficient mapping of 3D points into compact memory spaces. Benefiting from compact memory layouts, Flash3D introduces a multilevel attention grouping method, **Bucket-and-Swin**, precisely aligned with GPU tiling, allowing zero overhead region shifting fused with FlashAttention-2 [6]. The joint locality and efficiency boost are illustrated in Figure 1.

Our key contributions are summarized as follows:

- **Unified geometric and GPU memory locality**: Using Perfect Spatial Hashing (PSH), we propose a principled locality mechanism that brings together geometric and memory locality, benefiting downstream feature quality and operation throughput.

- **Attention aligned with GPU tiling**: We propose a novel **Bucket-and-Swin** attention mechanism, structured closely to GPU tiling, incorporating zero-overhead region shifting and striding.
- **Scalable Performance**: Flash3D Transformer outperforms PTv3 on benchmark datasets, simultaneously achieving a 2.25x speed increase and 2.4x memory efficiency boost.

2. Related Work

3D Point Transformers Many variations of Point Transformers have appeared recently, such as PTv1 [46], PTv2 [40], PTv3 [41], FlatFormer [22], and OctFormer [39], each differing in their definitions of geometric locality. PTv1 and PTv2 introduced foundational concepts in point cloud transformers, focusing on capturing both local and global geometric structures within point clouds through intricate windowing of MHSA.

FlatFormer [22] demonstrated that partitioning point clouds into equally sized windows benefits MHSA by enabling efficient batching. OctFormer [39] proposed octree-based neighborbood partitioning for fast window computation. Additionally, region shifting based on Swin Transformers has been utilized to aggregate and distribute global information through subsequent local windowing, enhancing the network's ability to capture both local and global features [9, 21, 22]. PTv3 [41] incorporated such design principles and achieved state-of-the-art performance by scaling up MHSA window sizes and parameter sizes. However, by relying on abstractions of global serialization and scattering of points and features, PTv3 overlooks GPU architectural issues, incurring substantial and unnecessary computational and memory costs (see Figure 2 and Section 3 for details). This highlights the need to holistically integrate geometric locality with hardware considerations.

Varied definitions of geometric locality call for a general approach to encompass a family of such. Ideal geometric locality definitions should be flexible and compute-efficient to represent striding and shifting and incorporate global structures within point clouds. Variations among partition sizes lead to negligible differences in downstream task performances while hindering compute efficiency [36, 39]. Therefore, we motivate our principled locality mechanism to align flexible geometric locality definitions with evenly sized windows, which allow regularly batched computations on GPUs and saturate GPU compute throughput. Our **Bucket-and-Swin** attention offers an efficient method for striding and shifting on top of properly localized points by leveraging our principled locality mechanism.

FlashAttention FlashAttention algorithms [6, 7, 32] leverage the tiling structure of GPU chips and optimize attention mechanisms by retaining intermediate results on-chip and carefully localizing computations. Retention of intermediate results on-chip helps FlashAttention algorithms to reduce quadratic memory cost down to a linear one [7, 25, 33, 34]. Partitioning input arrays into tiles aligned with GPU tiling structures maximizes computation throughput. Given that GPU architectures have maintained a tiling structure for over two decades—a fundamental design aspect unlikely to change [13]—these methods aim to maximize GPU utilization for a range of GPU chips.

These two principles motivate our design choices of **Bucket-and-Swin** attention. We partition point cloud features into tiles fit in GPU tiles; each tile is fetched on-the-fly based on logical assignments; logical partitions and fetching are fused with FlashAttention-2 CUDA kernel.

Perfect Spatial Hashing (PSH) Spatial hashing was originally proposed to efficiently locate and query spatially sparse data points [12, 23]. To enhance memory locality and computational efficiency, researchers introduced the concept of *perfectness* [5], compacting the hash table into a contiguous linear array in memory. This approach resulted in improved memory access patterns and query speed [20]. Later, [1] proposed a parallelized PSH construction on GPUs, achieving exceptionally fast constructions even on early GPUs by leveraging their massive parallel processing capabilities and placing buckets within GPU tiles.

In our work, we focus on perfectness, propose a principled locality mechanism, and remove the traditional table indexing structure. We focus on hashing a large number of points into a contiguous memory array for MHSA. This simplification aligns with our goal of unifying geometric locality with GPU memory locality to optimize performance.

3. Preliminaries

In order to explain how Flash3D achieves fast performance, we must first understand the bottlenecks in previous work. Figure 2 depicts the high-level dataflow associated with a method such as PTv3 (note that some layers are omitted for the sake of simplicity of exposition). Each numbered rectangle represents a float stored in memory. We assume the data has been pre-sorted to ensure contiguous floats are likely to be close in terms of some meaningful metric.

Before the data can be processed, it must be loaded from main memory into the L1 cache (which is practically equivalent to shared memory in GPU jargon). An H100 can perform this step at a rate of roughly 1 TFloat/s. Although this seems like an impressive number, it is dwarfed by the maximum achievable FLOP throughput of the GPU, which is in excess of 60 TFloat/s. In our case, the compute-intensive step consists of FlashAttention, which can perform in excess of 500 TFLOPs/s (albeit at reduced precision).

After writing results to memory, PTv3 then performs a global reordering of the points in order to communicate in-

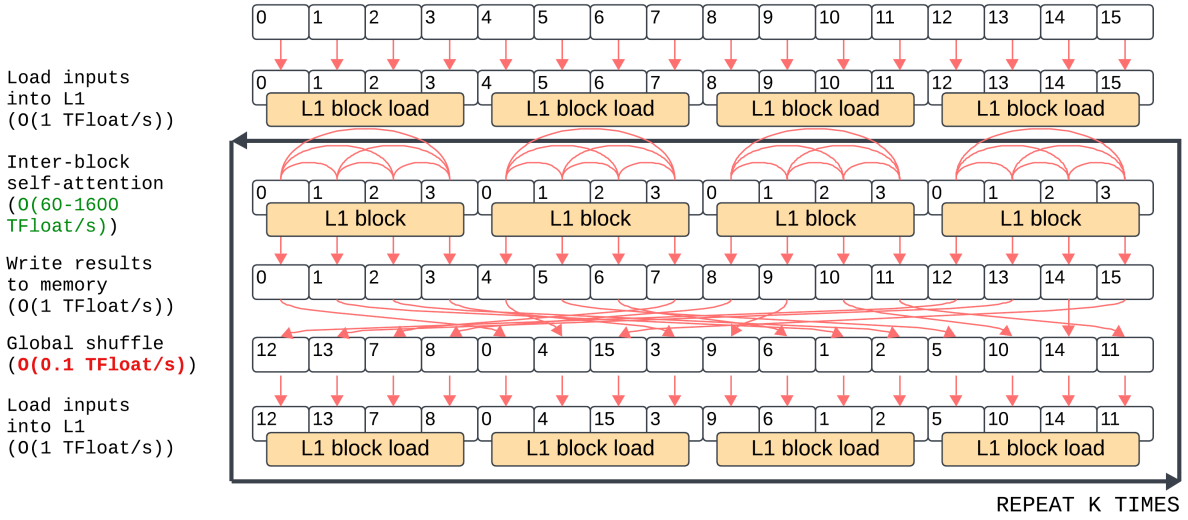


Figure 2. High-level schematic overview of PTv3. Numbered rectangles represent locations in memory. Adjacent rectangles are adjacent in memory. Arrows indicate data movement. See Section 3 for details.

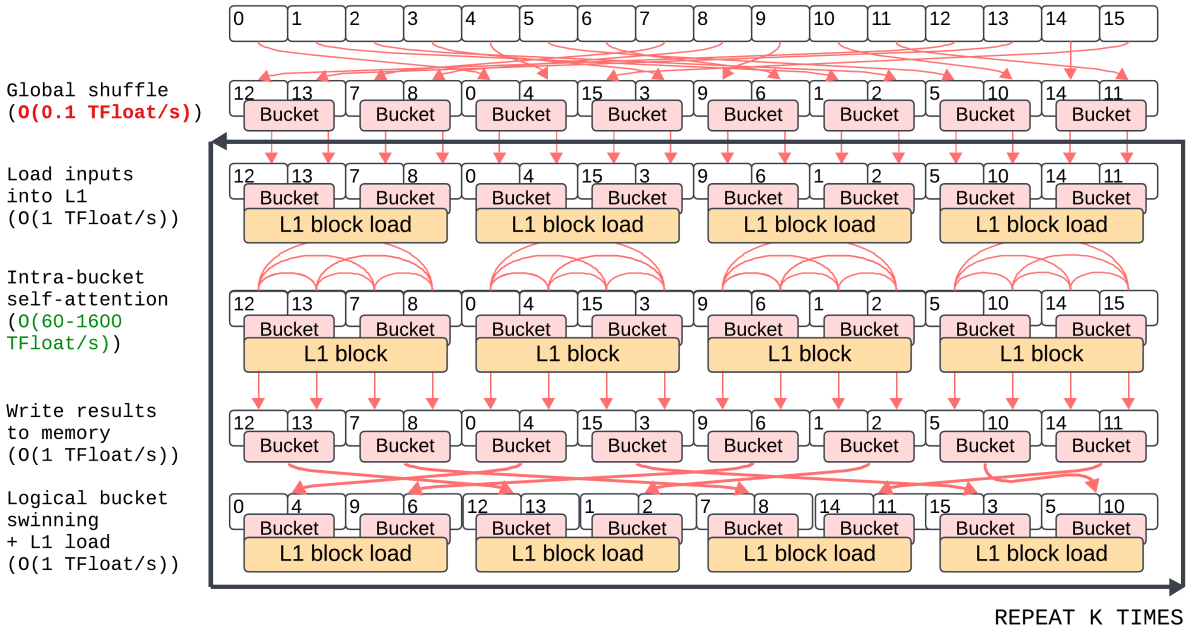


Figure 3. High-level schematic overview of Flash3D. Flash3D performs multiple rounds of attention with different neighborhood definitions via our *bucket-and-swin* approach, which saves an expensive global shuffle in each round. See Section 4 for details.

formation across a different set of neighborhoods. As depicted in the figure, this step is orders of magnitude slower than the other steps. This is because the GPU is optimized to transfer large amounts of data in transactions of contiguous blocks. Transferring data between random locations is analogous to using many buckets (memory transactions) to move water one drop (float) at a time—this process can only be efficient if we fill the buckets (by transferring the data in contiguous chunks)[1].

[1]Though global shuffling through raw DRAM channels bottlenecks at 0.1TFloat/s, Flash3D effectively coalesces shuffling in L2 cache and approximates 1TFloat/s limit in reality. One million points `FP16`$[1\text{M}, 3]$ cost 6MB memory while H100 has 50MB L2 cache to coalesce shuffled results, write well-packed results, and maximize DRAM bandwidth.

4. Method

From the preceding discussion, it is clear that the global shuffle step of PTv3 is a key bottleneck. Our key idea is to mitigate this bottleneck through our **Bucket-and-Swin** strategy. In this approach, illustrated in Figure 3, we initially bucket the points into spatially similar neighborhoods using hash functions. This involves an up-front global shuf-

fle operation that places points in the same contiguous block of memory if they are in the same bucket.

In subsequent iterations, instead of repeating a slow global shuffle in order to communicate information using different neighborhoods, we logically shuffle the buckets to yield different neighborhoods for attention. This shuffle is logical in the sense that it does not involve any permutation of bytes in memory—instead, we simply load the appropriate buckets into L1 just before calculating attentions. To return to the water analogy, this is akin to transferring water using full buckets of water (by loading entire point-buckets into L1) instead of nearly empty buckets. We are thus able to mostly mitigate the global shuffling bottleneck.

Implementing this strategy efficiently involves a few non-trivial details, which we elaborate upon in the following subsections.

4.1. Principled Locality Mechanism with PSH

Our principled locality mechanism leverages Perfect Spatial Hashing (PSH) [1, 20] to map spatially sparse 3D point clouds into compact and contiguous arrays. PSH constructs a bijection to scatter 3D points. In the output array, memory address proximity implies spatial proximity.

Input and Output The PSH algorithm takes as input the point coordinates $\mathbf{C} \in \mathbb{R}^{N\times 3}$, representing N points in 3D space, and the bucket capacity S, defining the maximum points per bucket. It outputs the bucket IDs $\mathbf{bucket\_id} \in \mathbb{Z}^N$, assigning each point to a bucket, and bucket offsets $\mathbf{bucket\_offset} \in \mathbb{Z}^N$, indexing each point's position within its assigned bucket. The index of the i-th point can be determined by $\mathbf{bucket\_base}[\mathbf{bucket\_id}[i]] + \mathbf{bucket\_offset}[i]$. When buckets are concatenated back-to-back, the resulting indices are guaranteed to be contiguous, and scattering points according to these indices produces a contiguous array of points. We describe our full PSH construction algorithm in Alg 1 and Alg 2.

Stage Orchestration In Flash3D, PSH orchestrates three key components within each transformer stage. First, PSH groups spatially adjacent points into contiguous memory arrays and produces $\mathbf{bucket\_id}$, $\mathbf{bucket\_offset}$. The resulted array suits point-wise MLPs and LayerNorm. The second component maps buckets to attention scopes during FlashAttention without extra cost. Lastly, the bucket structure directs a tile-level in-bucket pooling layer. Because points within each bucket are geometrically local, pooling operations are confined within buckets, reducing the need for costly global neighbor queries. This localized pooling achieves high computational efficiency by retaining geometric coherence in point clusters, enabling efficient, spatially-aware feature aggregation throughout the model.

Bucket rebalancing Ideally, we would assign a bucket to each point by simply hashing it. Unfortunately, this would lead to imbalanced buckets, with some buckets containing many points and others containing few points. Such non-uniform buckets would prevent us from efficiently mapping buckets onto GPU tiles. To avoid this issue, we aim for *best-effort* geometric locality instead of rigidly defining geometric boundaries, allowing the GPU to settle point-to-bucket assignments opportunistically. This declarative approach yields additional advantages.

Algorithm 1 Batch PSH Bucketing and Balancing

Require: Coordinates $\mathbf{C} \in \mathbb{R}^{N\times D}$, batch indices $\mathbf{B}$, number of buckets K, bucket capacity S
Ensure: Bucket IDs $\mathbf{bucket\_id}$, Bucket Offsets $\mathbf{bkt\_off}$

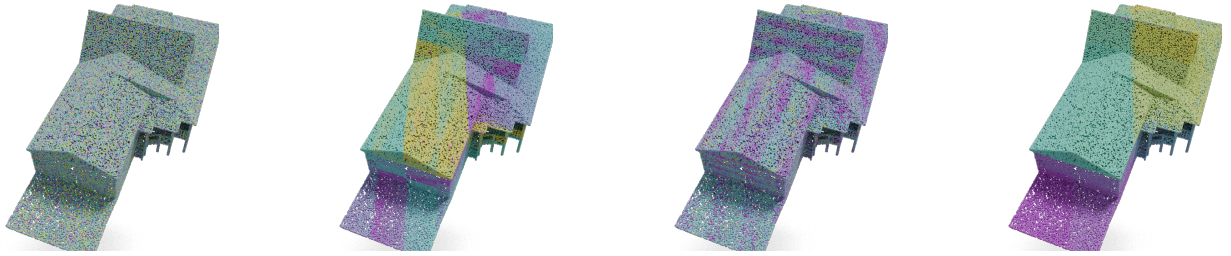
for each batch b **in parallel do**
  Initialize bucket counters $\mathbf{bkt\_ctr} \leftarrow 0$
  for each point i in batch b **in parallel do**
    Compute voxel coordinate $\mathbf{v}_i$ from $\mathbf{C}_i$
    Compute initial bucket ID h_i from $\mathbf{v}_i$ using hash function
    if $\mathbf{bkt\_ctr}[h_i] < S$ **then** ▷ Assign point i to bucket h_i
      $\mathbf{bucket\_id} \leftarrow h_i$
      $\mathbf{bkt\_off}[i] \leftarrow$ AtomicInc($\mathbf{bkt\_ctr}[h_i]$)
    else
      OptimisticRacing(i, $\mathbf{v}_i$, $\mathbf{bkt\_ctr}$, S)
    end if
  end for
  Perform exclusive scan on $\mathbf{bkt\_ctr}$ to compute $\mathbf{bucket\_base}$
  Scatter points into contiguous memory based on bucket assignments
end for

First, relaxed geometric boundaries allow point-bucket associations to be resolved in on-chip arbitration, significantly boosting grouping throughput. Second, softened boundaries result in diffused geometric patterns, facilitating more flexible feature extraction through attention mechanisms. Randomly perturbed boundaries regularizes attention mechanisms without costly feature shuffling as seen in PTv3 [41]. We show the effects in Table 3.

Hash functions Our principled locality mechanism in Flash3D leverages the flexibility of hash functions to define geometric locality. This approach benefits from a declarative structure, allowing geometric locality definitions to be managed directly through hash functions. In contrast to learning from data, our hash functions are manually defined to capture common geometric patterns effectively, as demonstrated by our empirical results.

In this work, we utilize four hash functions that operate over point coordinates to distribute spatially close points into buckets. These include **XOR-mod**, **XOR-div**, **Zorder-mod**, and **Zorder-div**:

XOR-mod: This function computes the XOR of the voxel



(a) XOR-mod (Rebalanced) (b) XOR-div (Rebalanced) (c) Zorder-mod (Rebalanced) (d) Zorder-div (Rebalanced)

Figure 4. Illustration of bucket assignments using four hash functions after rebalancing. Colors of points indicate their bucket assignments. We demonstrate our PSH algorithm on a sample point cloud with 100k points from BuildingNet [31].

Algorithm 2 Optimistic Racing for Bucket Reassignment

Require: Point index i, voxel coordinate $\mathbf{v}_i$, bucket counters $\mathbf{bkt\_ctr}$, bucket capacity S
Ensure: Updated bucket assignment for point i

```
Initialize assigned bucket ID h_assigned ← −1
for each probe offset δ do
    Perturb voxel coordinate v'_i ← v_i + δ
    Compute new bucket ID h'_i from v'_i using hash function
    if bkt_ctr[h'_i] < S then
        prev_off = AtomicInc(bkt_ctr[h'_i])
        if prev_off ≤ S then                ▷ Confirm bucket capacity after increment
            Assign point i to bucket h'_i
            bucket_offset[i] ← prev_off
            h_assigned ← h'_i
            break
        else
            AtomicDec(bkt_ctr[h'_i])        ▷ Roll back if over capacity
        end if
    end if
end for
if h_assigned = −1 then
    Assign point i to recycle bucket r
    bucket_offset[i] ← AtomicInc(bkt_ctr[r])
end if
```

coordinate bits and applies a modulo operation. Given voxel coordinate $\mathbf{v} \in \mathbb{Z}^3$, the XOR-mod hash is defined as:

$$h(\mathbf{v}) = \left(\bigoplus_{d=1}^{3} v_d \right) \mod K$$

where v_d represents the d-th dimension of $\mathbf{v}$, and K is the number of buckets. XOR-mod evenly spreads the geometric overage of each bucket.

XOR-div: Similar to XOR-mod, this function computes the XOR of the voxel coordinate bits but uses division to determine the bucket. The XOR-div hash is defined as:

$$h(\mathbf{v}) = \left(\bigoplus_{d=1}^{3} v_d \right) \div S$$

where S is a scaling factor that controls the bucket size. XOR-div ensures uniform bucket distribution across large voxel ranges, while keeping tight intra-bucket coherence.

Zorder-mod: This function calculates a Z-order (Morton) [26] code by interleaving the bits of each dimension in $\mathbf{v}$, then applies a modulo operation:

$$h(\mathbf{v}) = Z(\mathbf{v}) \mod K$$

where $Z(\mathbf{v})$ represents the interleaved Z-order code of $\mathbf{v}$. Zorder-mod preserves spatial proximity, mapping nearby points to similar buckets.

Zorder-div: This function also uses Z-order coding but divides the Z-order value to allocate points to buckets:

$$h(\mathbf{v}) = Z(\mathbf{v}) \div S$$

where S is a divisor that adjusts the bucket density. Zorder-div is effective for capturing structured spatial locality over extensive regions.

These hash functions provide the necessary flexibility to define geometric locality without the complexity of learning from data, making them both efficient and versatile. Additional hash functions can be defined as needed, allowing Flash3D to adapt to diverse geometric patterns and data distributions.

4.2. Flash3D Transformer Stage

Windowed attention mechanisms are well-suited to point clouds, as they capture both local and global features by focusing on spatially close points and shifting windows across regions [9, 22, 40, 41]. We describe how Flash3D achieves local and global feature extraction with minimal computational costs.

Zero-Overhead Bucket-Swin Attention Flash3D introduces a bucket-based Swin attention mechanism that partitions point features into spatially coherent buckets, enabling windowed attention shifts at zero additional cost. This design maximizes memory locality and facilitates efficient intra- and inter-bucket feature propagation. Given

a feature collection $\{\mathcal{F}_{\mathcal{A}_i}\}$ over an attention scope i, the mechanism is defined as:

$$\mathcal{F}'_{\mathcal{A}_i} = \mathrm{MHSA}\Big(\mathrm{LN}\big(\{\mathcal{F}_{\mathcal{A}_i}\}\big),\, \mathrm{PE}(\mathbf{C}_{\mathcal{A}_i})\Big) + \mathcal{F}_{\mathcal{A}_i},$$

$$\tilde{\mathcal{F}}_{\mathcal{A}_i} = \mathrm{MLP}\Big(\mathrm{LN}(\mathcal{F}'_{\mathcal{A}_i})\Big) + \mathcal{F}'_{\mathcal{A}_i}.$$

For example, consider bucket indices:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$

An initial attention scope may cover buckets $\{1, 2, 3, 4\}$; a subsequent shift by two buckets yields $\{3, 4, 5, 6\}$, enabling cross-bucket interactions. This bucket-level organization permits Swin-style window shifting without global neighborhood recomputation, achieving memory- and computation-efficient zero-overhead shifts. Flash3D thus leverages the intrinsic geometric structure of point clouds to streamline attention operations and enhance feature extraction. We visualize **Bucket-Swin** attention scopes in Appendix 9.

In-bucket Pooling Conventional grid pooling in PTv2 [40] incurs significant inter-tile communication overhead. In contrast, our approach leverages geometric locality by scattering points into contiguous arrays such that each bucket-aligned subarray contains spatially proximate points that fit entirely within a GPU tile and its fast L1 cache (i.e., shared memory, see Section 3). This enables efficient intra-tile bucketing and balancing, outperforming our main PSH algorithm. Our in-bucket pooling layer employs a fixed reduction factor ρ (e.g., $\rho = 2$ for $2\times$ pooling) over 1024 points per tile, applying reduction operations (e.g., `sum`, `mean`, `min`, `max`) followed by a bulk memory transaction to minimize overhead. Unlike PTv3's globally sorted pooling, our method fully exploits both geometric and memory locality, thereby eliminating latency penalties. Further details on sub-bucket construction and balancing are provided in Appendix 10.5.

5. Experiments

We evaluate both *downstream task performance* and *scalability*, demonstrating the impact of design choices from Section 4 through task performance and ablation studies. In the subsequent section, we validate the hypotheses from Section 3 and detail the resulting scalability. Flash3D and baseline methods are benchmarked on server GPUs (A100, H100) and edge GPUs (L4), with evaluation metrics reported on a single GPU (A100, H100, L4) to ensure fair comparisons; training costs on H100 GPUs are provided in Appendix 8. Our fused **Bucket-and-Swin** attention is implemented using ThunderKittens [34], with implementation details in Appendix 10.2.

5.1. Indoor 3D Semantic Segmentation

We benchmark PTv3 and Flash3D on the ScanNet semantic segmentation task using NVIDIA A100 and L4 GPUs (see Tab. 1). On dense point clouds (500k–2m points per scene), Flash3D outperforms PTv3 at a fraction of its cost. PTv3's latency degrades significantly from A100 to L4 due to imbalanced workload distribution, whereas Flash3D preserves its latency advantage across hardware platforms.

Indoor Sem. Seg.	ScanNet				A100	L4
Methods	Val	Test	Params.	Memory	Latency	Latency
MinkUNet	72.2	73.6	37.9M	4.7G	90ms	-
PTv3	77.5	77.9	46.2M	5.2G	61ms	98ms
Flash3D	77.9	**78.6**	46.2M	**2.4G**	**24ms**	**35ms**

Table 1. Indoor scene benchmarks with edge GPUs

5.2. Outdoor 3D Semantic Segmentation

Scalability (nuScenes)	Params.	Memory	mIoU	A100 Latency	H100 Latency	L4 Latency
MinkUNet [4]	37.9M	1.7G	73.3	48ms	34.3ms	-
PTv3 [41]	46.2M	1.2G	80.4	45ms	30.2ms	69ms
Flash3D	46.2M	**0.5G**	**81.2**	**20ms**	**13.4ms**	**24ms**
PTv3 [41]	46.2M	1.2G	80.4	45ms	30.2ms	69ms
Flash3D	129.4M	1.2G	**81.5**	**24ms**	**15.1ms**	**29ms**

Table 2. Model scalability comparisons of MinkUNet, PTv3 and Flash3D on A100, H100, and L4 GPUs by fixing model parameter sizes and memory quotas respectively. Dark cells indicate fixed budgets. We fix attention scopes of all models at 4096. mIoU indicates the semantic segmentation performance on nuScenes validation set.

We benchmark Flash3D on the nuScenes semantic segmentation task [2, 10] against two prior state-of-the-art methods (see Tab. 2). Two Flash3D variants were trained under distinct quotas to assess scalability. In the first setting—equal parameter counts between Flash3D and PTv3 [41] (upper half of Tab. 2)—Flash3D achieves a 1.0% mIoU improvement and $2.25\times$ faster inference while using $2.4\times$ less memory. In the second setting—with fixed memory quotas (lower half of Tab. 2)—Flash3D accommodates $2.8\times$ more parameters and yields a 1.4% mIoU gain, with inference latency still $1.88\times$ faster. These results confirm Flash3D's robust scalability. For additional downstream benchmarks, see Appendix 7. We further present an ablation study on the nuScenes validation set.

Hashs	standard	+rebalance	+stride	+swin
XD	78.6	78.7	78.9	79.2
XD+ZD	79.1	79.1	79.8	80.6
XD+XM	78.9	78.9	78.9	79.4
XD+XM+ZD+ZM	79.3	79.4	80.2	**81.2**

Table 3. Hash Function Variants and Bucket-based strides and swin on the validation split of the nuScenes dataset. +REBALANCE adds bucket rebalancing and stochastic geometric boundary perturbations per Alg 2. +STRIDE strides attention scopes at two buckets. +SWIN shifts attention scopes at one bucket a time.

Hash Functions and Rebalancing We evaluate hash and stacked multi-hash scattering on semantic segmentation

(Tab. 3). When stacking transformer stages, a change in the hash function indicates a switch (adding a new hash), whereas using the same hash maintains a constant stage count. Notably, **XOR-div** serves as a robust baseline; combining it with a structural hash (e.g., **Zorder-div**) yields further gains, while pairing with an evenly distributed hash (e.g., **XOR-mod**) offers only marginal improvements. Integrating all hash functions defined in Section 4 attains peak performance. Additionally, PSH variants with and without the rebalancing step (Alg. 2) indicate that rebalancing exerts minimal effect on overall performance.

Stride and Swin We evaluate bucket striding and shifting (Swin) patterns in conjunction with our stacked multi-hash scattering (Tab. 3). Bucket strides expand receptive fields, yielding modest gains, while bucket shifting smooths attention transitions and further enhances performance.

5.3. Scalability Analysis

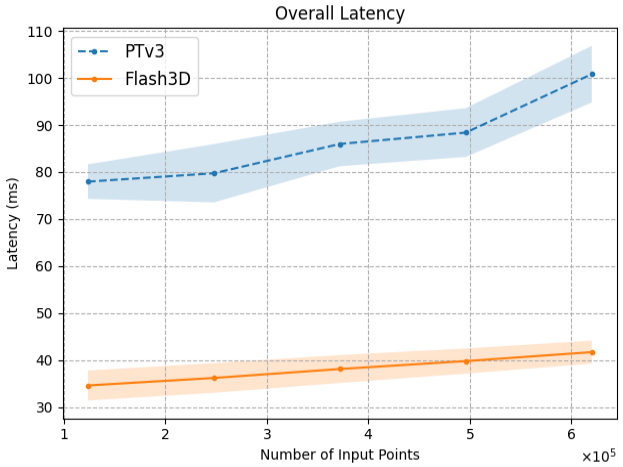


Figure 5. Overall Latencies vs. Input Sizes for Flash3D and PTv3.

As hypothesized in Section 1, scalable point transformers should respect *GPU tiling* and *memory locality*. In this subsection, we provide detailed profiling results on a single A100 GPU to analyze Flash3D's scalability based on these principles. Specifically, we benchmark Flash3D and PTv3 in terms of latency, compute utilization, and DRAM bandwidth utilization under various input sizes. For this subsection, we fix Flash3D and PTv3 at the same parameter size.

We vary input sizes from 100k points to 600k points to stress test Flash3D and PTv3. We run both backbones for 20 iterations for each configuration and input size and discard the first warm-up run. With NVIDIA Nsight Systems [28], we collect tracing logs to report the `averages` and `standard deviations` for each metric.

5.3.1 Latency

We measure the overall latencies and latency breakdowns in this section. We also show the significant latency difference between Flash3D's PSH-based principled locality mechanism and the serialization algorithm of PTv3.

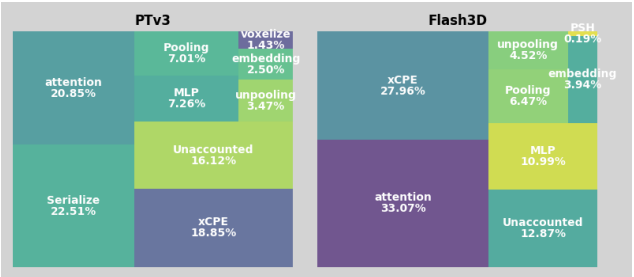


Figure 6. Latency TreeMap breakdowns for Flash3D and PTv3.

Overall Latency We increase the input sizes and show the overall latency difference of Flash3D and PTv3 in Figure 5. Flash3D consistently out-speeds PTv3 by more than 2x while keeping slower growth and narrower standard deviations. Flash3D carefully spreads workloads to GPU tiles without global serialization, so Flash3D enjoys a slow linear growth. On the other hand, global serialization and sorting algorithms in PTv3 incur super-linear complexity and exhibit super-linear latency costs.

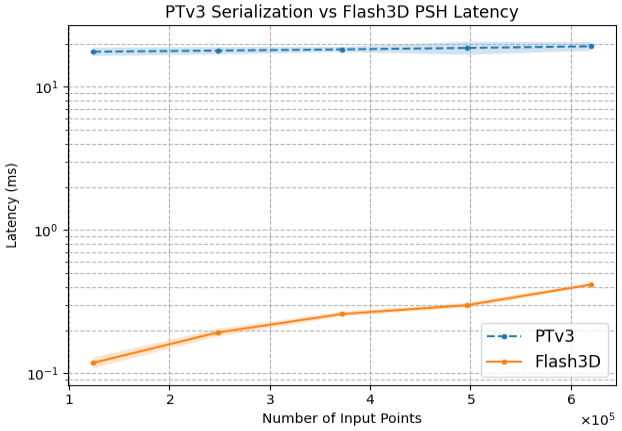


Figure 7. PTv3 Serialization vs. Flash3D PSH, log scale.

Latency Breakdown with Fixed Input Size To further understand the sources of latency, we perform a detailed breakdown of the time spent in different components of Flash3D and PTv3, shown in Figure 6. The global serialization costs a major portion of PTv3's latency while Flash3D's PSH latency cost is negligible (0.19%).

Serialization vs PSH Latency We isolate the latencies of serialization and Flash3D PSH in Figure 7. Flash3D PSH latency is consistently two orders of magnitude lower than PTv3 serialization. This difference is a powerful proof of scalability impacts by respecting *GPU tiling*.

5.3.2 Hardware Utilization Analysis

We provide an in-depth analysis of hardware utilization to evaluate how efficiently Flash3D utilizes GPU resources during execution. We focus on three key metrics: compute

utilization, matrix multiplication utilization, and memory bandwidth utilization.

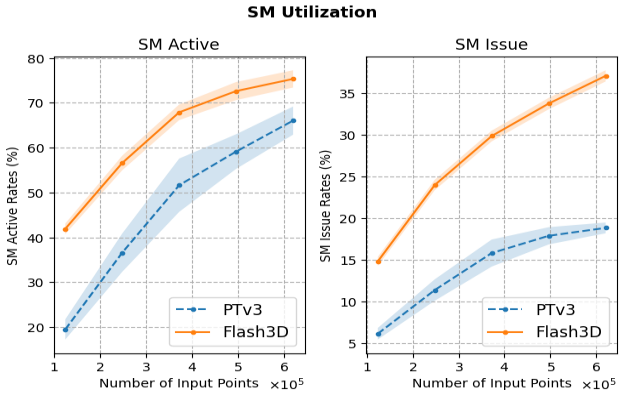


Figure 8. SM Utilization vs. Input Sizes for Flash3D and PTv3. We show the overall SM active rates on the left and more specific SM issuing rates on the right.

General Compute Utilization We report the SM (Streaming Multiprocessor) active rates and SM issuing rates in Figure 8. Both measure GPU tile utilization while SM issuing rates are more specific to warp dispatcher instruction throughput.

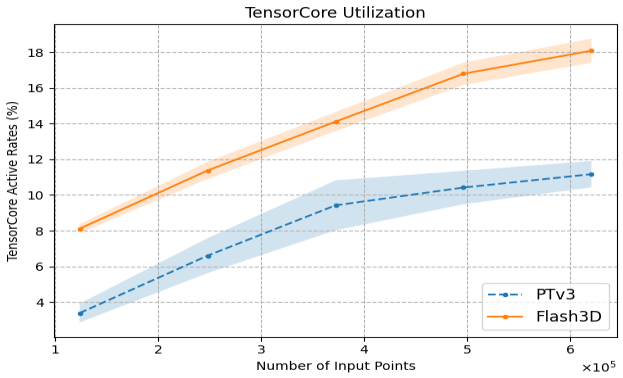


Figure 9. TensorCore Active Rates vs. Input Sizes for Flash3D and PTv3.

TensorCore Matrix Multiplication Utilization TensorCores contribute an overwhelming computing throughput to modern GPU chips [2]. TensorCore saturation improvements have amplified impacts on overall throughputs. PTv3 struggles with less than 5% TensorCore utilization in most point cloud scenarios, **wasting over 95%** of GPU resources and investments. In contrast, Figure 9 highlights Flash3D overcomes these bottlenecks.

Memory Bandwidth Utilization (DRAM Read) Finally, we evaluate memory read bandwidth utilization (Figure 10). Flash3D excels in memory bandwidth usage metric by respecting *memory locality*. PTv3 bottlenecks on memory bandwidth due to its multiple global scattering operations.

[2]For H100 GPUs, FP16 TensorCores account for 1979 teraFLOPS among total peak computing capacities while traditional FP32 CUDA Cores account for 67 teraFLOPS.

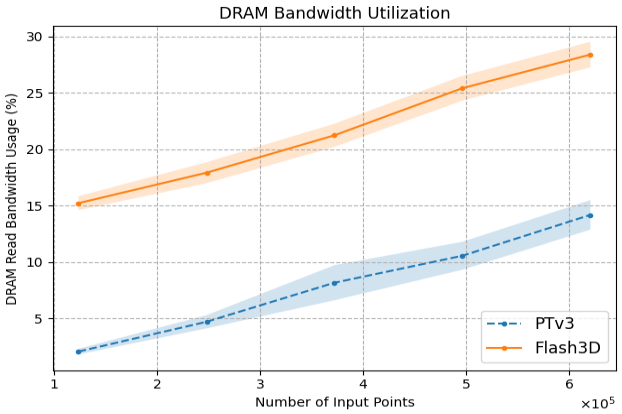
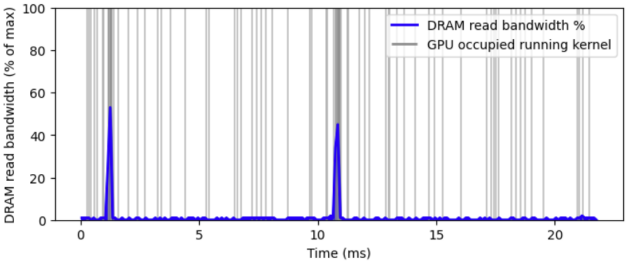


Figure 10. DRAM Read Bandwidth Usage vs. Input Sizes for Flash3D and PTv3.

5.4. Shuffling Slowness Analysis

Figure 11 precisely corroborates our claim regarding the slowness of PTv3 shuffling. The x-axis, spanning the shuffling interval, is marked by gray vertical bars that denote minimal GPU occupancy, while persistently low memory bandwidth indicates severe underutilization. Although PTv3's shuffling could be marginally optimized, fundamental hardware constraints (cf. [19], Fig. 5) inherently bound scatter performance. These findings directly support our claim that a geometry-hardware co-design approach is essential for enhancing task performance and achieving overall scalability.

Figure 11. Tracing logs of PTv3 Serial Shufffling 120k points. In reality, PTv3 shuffling takes **738** CUDA kernels.



6. Conclusion and Future Work

Flash3D Transformer unifies algorithm design with hardware optimization, enabling rapid training and inference in point cloud backbones. By leveraging joint hardware-geometry locality, it achieves a 2.25× speedup and 2.4× memory efficiency over prior point transformers, underscoring the need for algorithm-hardware co-design on modern GPUs. While we recognize the challenges of integrating innovative methods into established pipelines, our work provides a clear, robust framework for adoption. Future research will explore model distillation, early sensor fusion, and edge deployment strategies on Orin and ThorU (see Appendix 11). We conclude that Flash3D offers a practical and transformative solution, paving the way for scalable, high-performance point cloud processing.

References

[1] Dan A Alcantara, Andrei Sharf, Fatemeh Abbasinejad, Shubhabrata Sengupta, Michael Mitzenmacher, John D Owens, and Nina Amenta. Real-time parallel hashing on the gpu. In *ACM SIGGRAPH asia 2009 papers*, pages 1–9. 2009. 2, 4

[2] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 6, 1

[3] Ran Cheng, Ryan Razani, Ehsan Taghavi, Enxu Li, and Bingbing Liu. 2-s3net: Attentive feature fusion with adaptive feature selection for sparse semantic segmentation network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12547–12556, 2021. 1

[4] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 6, 1, 7

[5] Zbigniew J Czech, George Havas, and Bohdan S Majewski. Perfect hashing. *Theoretical Computer Science*, 182(1-2): 1–143, 1997. 2

[6] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023. 1, 2, 4, 5

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022. 2, 4

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 7

[9] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8458–8468, 2022. 2, 5, 1

[10] Whye Kit Fong, Rohit Mohan, Juana Valeria Hurtado, Lubing Zhou, Holger Caesar, Oscar Beijbom, and Abhinav Valada. Panoptic nuscenes: A large-scale benchmark for lidar panoptic segmentation and tracking. *IEEE Robotics and Automation Letters*, 7(2):3795–3802, 2022. 6, 1

[11] Anton Franzluebbers, Changying Li, Andrew H. Paterson, and Kyle Johnsen. Virtual reality point cloud annotation. *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 886–887, 2022. 1

[12] Michael L Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0 (1) worst case access time. *Journal of the ACM (JACM)*, 31(3):538–544, 1984. 2

[13] John L. Hennessy and David A. Patterson. *Computer Architecture, Sixth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6th edition, 2017. 2

[14] Xiangwang Hu, Zuduo Zheng, Danjue Chen, Xi Zhang, and Jian Sun. Processing, assessing, and enhancing the waymo autonomous vehicle open dataset for driving behavior research. *Transportation Research Part C: Emerging Technologies*, 134:103490, 2022. 1

[15] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862, 2023. 1

[16] Leela S Karumbunathan. Nvidia jetson agx orin series. *Online at https://www. nvidia. com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief. pdf*, 2022. 7

[17] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified transformer for 3d point cloud segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8500–8509, 2022. 1

[18] Xin Lai, Yukang Chen, Fanbin Lu, Jianhui Liu, and Jiaya Jia. Spherical transformer for lidar-based 3d recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17545–17555, 2023. 1

[19] Patrick Lavin et al. Evaluating gather and scatter performance on cpus and gpus. In *Proceedings of the International Symposium on Memory Systems*, 2020. 8

[20] Sylvain Lefebvre and Hugues Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics (TOG)*, 25(3):579–588, 2006. 2, 4

[21] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021. 1, 2

[22] Zhijian Liu, Xinyu Yang, Haotian Tang, Shang Yang, and Song Han. Flatformer: Flattened window attention for efficient point cloud transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1200–1211, 2023. 1, 2, 5

[23] Ming-Ling Lo and Chinya V Ravishankar. Spatial hash-joins. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 247–258, 1996. 2

[24] Kenzo Lobos-Tsunekawa and Tatsuya Harada. Point cloud based reinforcement learning for sim-to-real and partial observability in visual navigation. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5871–5878, 2020. 1

[25] Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018. 2

[26] Guy M Morton. A computer oriented geodetic data base and a new technique in file sequencing. 1966. 5

[27] Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, David J. Weiss, Benjamin Sapp, Zhifeng Chen, and Jonathon Shlens. Scene transformer: A unified multi-task model for behavior prediction and planning. *ArXiv*, abs/2106.08417, 2021. 1

[28] CUDA Nvidia. Nvidia nsight systems 2024.6.1. *NVIDIA: Santa Clara, CA*, 2024. 7

[29] Yuzhe Qin, Binghao Huang, Zhao-Heng Yin, Hao Su, and Xiaolong Wang. Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation. In *Conference on Robot Learning*, 2022. 1

[30] Shi Qiu, Saeed Anwar, and Nick Barnes. Semantic segmentation for real point cloud scenes via bilateral augmentation and adaptive fusion. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1757–1767, 2021. 1

[31] Pratheba Selvaraju, Mohamed Nabail, Marios Loizou, Maria Maslioukova, Melinos Averkiou, Andreas Andreou, Siddhartha Chaudhuri, and Evangelos Kalogerakis. Buildingnet: Learning to label 3d buildings. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 10377–10387. IEEE, 2021. 5

[32] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608*, 2024. 2, 4

[33] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3531–3539, 2021. 2

[34] Benjamin F Spector, Simran Arora, Aaryan Singhal, Daniel Y Fu, and Christopher Ré. Thunderkittens: Simple, fast, and adorable ai kernels. *arXiv preprint arXiv:2410.20399*, 2024. 2, 6, 4

[35] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 1

[36] Pei Sun, Mingxing Tan, Weiyue Wang, Chenxi Liu, Fei Xia, Zhaoqi Leng, and Dragomir Anguelov. Swformer: Sparse window transformer for 3d object detection in point clouds. In *European Conference on Computer Vision*, pages 426–442. Springer, 2022. 2

[37] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *European conference on computer vision*, pages 685–702. Springer, 2020. 1

[38] Halil Ibrahim Ugurlu, Huy Xuan Pham, and Erdal Kayacan. Sim-to-real deep reinforcement learning for safe end-to-end planning of aerial robots. *Robotics*, 11:109, 2022. 1

[39] Peng-Shuai Wang. Octformer: Octree-based transformers for 3d point clouds. *ACM Transactions on Graphics (TOG)*, 42(4):1–11, 2023. 1, 2

[40] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. *Advances in Neural Information Processing Systems*, 35:33330–33342, 2022. 2, 5, 6, 1

[41] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4840–4851, 2024. 1, 2, 4, 5, 6

[42] Xinchen Yan, Mohi Khansari, Jasmine Hsu, Yuanzheng Gong, Yunfei Bai, Sören Pirk, and Honglak Lee. Data-efficient learning for sim-to-real robotic grasping using deep point cloud prediction networks. *ArXiv*, abs/1906.08989, 2019. 1

[43] 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

[44] Tengju Ye, Wei Jing, Chunyong Hu, Shikun Huang, Lingping Gao, Fangzhen Li, Jingke Wang, Ke Guo, Wencong Xiao, Wei Mao, Hang Zheng, Kun Li, Junbo Chen, and Kaicheng Yu. Fusionad: Multi-modality fusion for prediction and planning tasks of autonomous driving. *ArXiv*, abs/2308.01006, 2023. 1

[45] Uksang Yoo, Hanwen Zhao, Alvaro Altamirano, Wenzhen Yuan, and Chen Feng. Toward zero-shot sim-to-real transfer learning for pneumatic soft robot 3d proprioceptive sensing. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 544–551, 2023. 1

[46] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021. 2

[47] Hui Zhou, Xinge Zhu, Xiao Song, Yuexin Ma, Zhe Wang, Hongsheng Li, and Dahua Lin. Cylinder3d: An effective 3d framework for driving-scene lidar semantic segmentation. *arXiv preprint arXiv:2008.01550*, 2020. 1