# Dynamic Sparse Voxel Attention for Efficient Transformers

Manel Bermad
Department of Computer Science
Stanford University
mbermad@stanford.edu

Veljko Skarich
Department of Computer Science
Stanford University
vskarich@stanford.edu

## Abstract

*Recent advances in 3D point-cloud classification have shown that hybrid point-voxel architectures can effectively leverage both local geometric detail and global context. However, existing voxel-based transformers such as Point-Voxel Transformer (PVT) rely on fixed-size sliding windowed attention over a dense voxel grid. In this paper, we propose Dynamic Sparse Dynamic Voxel Attention (DSVA), a mechanism that dynamically selects and attends over non-empty voxel tokens. We voxelize input points into a voxel grid, then filter empty voxels, compute local k-nearest-neighbor relationships, and finally learn a data-driven edge-scoring function that selects the top-k most salient neighbors. We integrate DSVA into the PVTConv backbone (formerly using 3×3×3 fixed-size sliding windows), resulting in a drop-in DSVA block without altering PVT's global point-cloud attention. While we came close to matching the SOTA performance of PVT's fixed-size window attention on ModelNet40, on the noisier ScanObjectNN our DSVA-enhanced network achieves consistent gains in classification accuracy.*

## 1. Introduction

Transformer self-attention on 3D data has a wide range of real-world applications, but the computational cost of processing point clouds or voxels is an order of magnitude higher than for 2D data. As full global self-attention for voxel transformers is computationally expensive, various mechanisms have been proposed to "sparsify" it. The general approach is to prune the token sequence by first selecting token pairs that are most likely to matter, then only compute attention on those.

One model that implements a variant of sparse 3D attention is called PVT: Point-Voxel Transformer for Point Cloud Learning [25], and it currently sits near the top of the accuracy rankings (93.7% no-voting reproduced) for pure transformer-based models on the ModelNet40 for 3D object classification. It uses sparse local attention by breaking up the voxel sequence into fixed-size sliding windows, where tokens self-attend only within each window, using vanilla dot-product attention. We use PVT as our baseline, and we propose to replace its fixed window attention mechanism with dynamic voxel windows that are based on input features. We call this mechanism Dynamic Sparse Voxel Attention (DSVA).

| Model → | Dynamic Sparse Voxel Attention (DSVA) |
|---|---|
| Type | Voxel Transformer |
| Inputs | Point Clouds |
| Outputs | 3D object class labels |
| Project Mods | Replace PVT's Window Attention with:<br>(1) spatial $k$-NN proposals<br>(2) MLP scoring on $[h_i \| h_j \| (p_i - p_j)]$<br>(3) top-$\tilde{k}$ masked Cross Attn. |
| Pre-training | PVT weights pretrained on ModelNet40 |
| Datasets | ModelNet40, ScanObjectNN |
| Metrics | OA, Precision, Recall |

Table 1. Project Summary

## 2. Related Work

To the best of our knowledge, no prior work dynamically scores voxels within local attention windows and selectively retains only the top-$k$ salient voxels during attention computation.

### 2.1. Voxel-Based Transformer Architectures

Voxel-based methods convert point clouds into regular grids for efficient sparse processing. Sparse Voxel Transformer [2] uses static sparsity within local windows—attending equally to all occupied voxels—without learned ranking or pruning. VoxelNeXt [4] relies on sparse convolutional encoders in a hierarchical cascade, foregoing Transformer attention or data-driven sparsification. VoTr [16] applies self-attention over non-empty voxels in fixed 3D windows (with dilated patterns) but cannot prioritize the most informative voxels. PVT [25] alternates local "Sparse

Window Attention" on voxels with a global point-branch, yet still treats all voxels in a window equally, leaving token selection unaddressed.

### 2.2. Point-Based Architectures

Point-based approaches operate on raw point clouds without voxelization. PointNet++ [14] aggregates multi-scale local neighborhoods via farthest-point sampling and PointNet modules, preserving fine detail but lacking long-range context. PointNet++ exemplifies the point-based methods category by capturing fine-grained local geometry via hierarchical grouping, but it does not include any explicit global attention mechanism.

Much like PointNet++ defines centroids via farthest-point sampling and then uses k-NN to form local neighborhoods, DSVA also relies on k-NN to propose candidate neighbors in voxel space. However, instead of treating all neighbors equally, DSVA passes each pair of voxel embeddings and relative coordinates through a small MLP to compute an importance score. In this way, both methods move beyond fixed grid-based receptive fields.

DGCNN [21] builds a dynamic k-NN graph per layer and applies EdgeConv, adapting neighborhoods as features evolve—yet graph construction and processing remain costly, and global relationships are underrepresented.

### 2.3. Graph-Based Attention Methods

GAT [19] extends self-attention to arbitrary graphs by learning attention coefficients $\alpha_{ij}$ for each node $i$ and neighbor $j$, dynamically weighting neighbor features; however, it relies on static graph connectivity (unless recomputed) and requires multiple layers to propagate global information, limiting efficiency for large point clouds.

### 2.4. Transformer-Based Point-Cloud Models

Transformer-based point models employ self-attention for long-range dependencies. PCT [7] embeds points via farthest-point sampling and groups with k-NN, using "offset-attention" (dependent on coordinate offsets) for translation robustness, but incurs $\mathcal{O}(N^2)$ attention cost. Point Transformer [27] introduces learnable relative positional encodings into attention over k-NN neighbors, achieving state-of-the-art performance but suffering similar quadratic cost and repeated k-NN overhead.

### 2.5. Sparse and Efficient Attention Schemes

To mitigate $\mathcal{O}(N^2)$ costs, sparse/approximate attention has been proposed. Shaw *et al.* [15] add learnable relative positional embeddings $\mathbf{a}_{ij}$ to full attention scores:

$$\text{Attention}(q_i, k_j) = \frac{q_i^\top k_j + q_i^\top \mathbf{a}_{ij}}{\sqrt{d}},$$

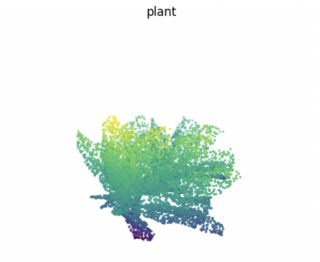


Figure 1. Example from ModelNet40

which boosts expressivity without reducing pairs. DSA [10] learns a sparsity predictor from low-rank query/key projections to mask out all but the top-$k\%$ of salient pairs, achieving up to 95% runtime sparsity with minimal accuracy loss—extending DSA to 3D requires adapting positional encoding and masking to voxel neighborhoods.

### 2.6. Spatio-Temporal Factorization

Inspired by video Transformers, TimeSformer [1] factorizes attention into separate spatial (within-frame patches) and temporal (across frames) modules, matching 3D CNN performance with lower cost. Its two-stage factorization parallels PVT's local voxel windows plus global point branch, but it uses fixed windows and does not dynamically select tokens within spatial regions.

### 2.7. Summary and Open Gap

Point-based methods [14, 21] capture fine-grained local geometry but lack explicit global attention. Transformer-based point models [7, 27] model global context with learnable positional biases yet incur $\mathcal{O}(N^2)$ cost. Sparse attention schemes [15, 10] add positional context or prune pairs but have not targeted voxel-level windowed attention. Voxel-based Transformers [2, 16, 4, 25] leverage grid structure and windowed attention but use static sparsity, treating all tokens equally within each window. None dynamically *rank* and *select* top-$k$ voxels per window based on input saliency. Our dynamic sparse voxel attention addresses this by learning to score and retain only the most informative voxel tokens in each window, merging windowed Transformer efficiency with adaptive sparsification.

## 3. Dataset and Features

As mentioned in our proposal, we are primarily using ModelNet40 [24], a collection of 3D CAD models for objects. We chose this dataset because ModelNet40 is the main synthetic dataset for 3D object classification, and almost every new 3D architecture reports results on it. It contains 40 classes, with 12k training examples and 2.5k test examples (see Fig. 1). Each example contains about 1k points / voxels. This dataset is suitable for a general development and testing


Figure 2. Examples from ScanObjectNN

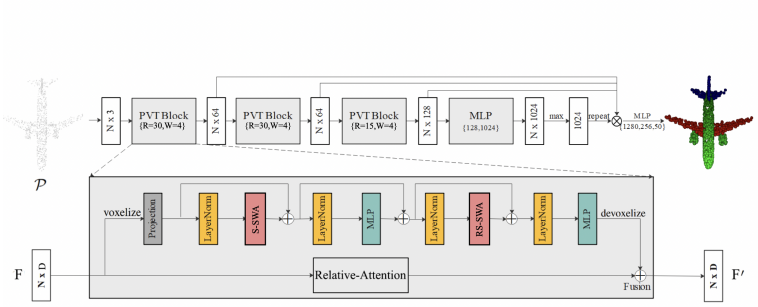


Figure 3. PVT Architecture

of new 3D architectures because every example is a complete, noise-free CAD model and has minimal confounding factors from sensor noise or missing data.

However, after using ModelNet40 to develop our approach, we test on a second, noisier dataset that can better demonstrate dynamic sparse attention's adaptive mechanism. In ModelNet40 all classes are sampled uniformly across their surfaces. This hides challenges of non-uniform density that dynamic sparse attention could address. Furthermore, every point belongs to the object. There are no outlier points or "clutter", so we would not be able to fully test voxel neighborhood selection under measurement uncertainty, nor the ability of the scoring MLP to ignore irrelevant voxels.

To address these concerns, we use the [17] ScanObjectNN dataset for secondary performance evaluation (see Fig. 2). It contains 15k real-scan objects in 15 classes, captured with RGB-D sensors, and the data contains background clutter, occluded objects and noise. To keep data processing uniform we computed surface normals for ScanObjectNN to match the built-in normals in ModelNet40.

# 4. Methods

## 4.1. PVT Architecture

The core inspiration behind PVT's architecture lies in reconciling two complementary paradigms for processing 3D data: the structured inductive biases of convolutional operations and the flexible, content-driven connectivity of self-attention. Convolutional layers excel at capturing local geometric patterns in a translation-equivariant way. A 3×3×3 or 5×5×5 convolution on a voxel grid naturally embeds the assumption that nearby voxels share meaningful context, and it guarantees that a learned filter will respond identically to the same feature regardless of where it appears in space. This built-in locality bias reduces the parameter count needed to learn low-level edge and surface primitives, accelerates convergence on finite datasets, and discourages overfitting by constraining interactions to a fixed neighborhood.

However, Transformer attention mechanisms introduce the possibility of arbitrarily large receptive fields. Instead of convolving a fixed kernel over all voxel activations, self-attention lets each query attend to potentially all keys, allowing the network to learn that distant but semantically related regions should exchange information. In 3D data where long-range context can matter, attention can capture those dependencies more directly than stacked convolutions might at affordable depth.

Because PVT's blocks alternate between these local-bias convolutions and more global attention, the network constructs a hierarchy of features: early layers learn simple geometric cues and small edges while later layers assemble these cues into higher-level object parts. This directly governs the progression of receptive fields: fewer, deeper layers with larger voxels yield coarser, more semantically abstract representations, whereas more, shallower layers with finer voxel grids specialize in fine-grained geometric detail.

However, stacking too many Transformer layers without sufficient inductive bias risks overfitting, especially on relatively small 3D datasets because the model might learn spurious long-range correlations that fail to generalize. However, by embedding sparse convolutional encoders at each resolution, PVT implicitly regularizes itself: small kernels must first carve out robust local relationships, and attention can only propagate what's already distilled into those local features.

As quantizing into voxels discards some positional precision, PVT re-injects point coordinates into each voxel token via a small coordinate MLP. This ensures that fine-grained geometric nuances are preserved. In this way, PVT leverages the best of both worlds: it inherits the inductive bias of locality from convolutions while preserving the ability to capture long-range relationships.

## 4.2. Extending PVT with Dynamic Sparse Attention

Dynamic sparse attention extends PVT by replacing its grid-aligned sliding attention windows with learned neighbor

selection that better reflects the irregular relationships in point-cloud scenes.

This design builds on PVT's hierarchical voxel encoding and local inductive bias but relaxes the assumption of uniform locality. In doing so, dynamic sparse attention preserves PVT's translation-equivariant features at small scales but adapts the receptive field based on feature simil

### 4.3. Training on ModelNet40 vs. ScanObjectNN

**Data Augmentation** For ModelNet40's uniformly sampled CAD meshes, augmentations are limited to yaw rotations, slight scaling, and minor jitter. Training uses SGD with momentum 0.9, weight decay 0.0001, step-decay or cosine schedules (initial LR 0.01–0.05), dropout 50 percent, and attention-dropout 10–20 percent; label smoothing is omitted. ScanObjectNN contains real-world scans with arbitrary orientations, occlusions, and sensor noise. Augmentations include full 3D rotations, random translations, broader scaling, and point dropout. Training uses AdamW $(LR = 0.002, \text{weight decay} 0.05)$ on a cosine-annealing schedule with warmup; regularization is stronger, with higher weight decay, more aggressive dropout, and label smoothing.

**DSVA Architecture** Each token $i$ first proposes a fixed set of $k$ candidate neighbors (e.g., via hashing or local windows), then computes a learned importance score

$$s_{ij} = \mathrm{MLP}\big([\, h_i \,\|\, h_j \,]\big)$$

for each candidate $j$, retains only the top-$k$ scored pairs per $i$, and finally runs vanilla dot-product multi-head self-attention

$$z_i = \sum_{j \in \mathcal{N}'(i)} \mathrm{softmax}\Big(\frac{q_i^\top k_j}{\sqrt{d_k}}\Big)\, v_j.$$

We first constructing a spatial $k$-NN graph over voxel centers $\{p_i\}$ to propose geometry-based neighbor candidates, then computing an importance score

$$s_{ij} = \mathrm{MLP}\big([h_i \,\|\, h_j \,\|\, (p_i - p_j)]\big)$$

for each edge and selecting the top-$\tilde{k}$ neighbors per token. After each voxel is assigned a neighborhood, we iterate across each anchor voxel and compute cross attention, where the anchor voxel acts as query. PVT contains 3 PVTConv blocks, the first of which operates on a $30^3$ voxel grid, and the other two at a $15^3$ resolution. We wanted each voxel's effective receptive field to remain commensurate with the original PVT resolutions, so for the fine-grained $30^3$ block we maintained a larger neighborhood and top-k neighbor group than the smaller $15^3$.

| Model | Data | Architecture | OA% |
|---|---|---|---|
| DSVA | SO | K-static, 1xD-Attn | **76.75** |
| Window | SO | Baseline | **76.34** |
| DSVA | SO | K-tuned,2xD-Attn | 74.01 |
| Window | MN | Baseline | **93.70** |
| DSVA | MN | K-tuned,2xD-Attn | **93.07** |
| DSVA | MN | K-static,1xD-Attn | 92.99 |
| DSVA | MN | No P-Attn,K-tuned,2xD-Attn | **91.37** |
| Window | MN | No P-Attn | 91.21 |
| DSVA | SO | No P-Attn,K-tuned,2xD-Attn,Reg | **73.04** |
| Window | SO | No P-Attn | 72.87 |
| DSVA | SO | No P-Attn,K-tuned,1xD-Attn | 72.76 |

Table 2. Summary of results, **K-tuned** = searching for best k-neighborhood and top-k values, **K-static** = k-neighborhood and top-k is same for all voxel resolutions, **P-Attn** = global point attention, **2xD-Attn** = increasing hidden dim 2x for attention, **Reg** = Aggressive 0.3 dropout.

## 5. Results

### 5.1. DSVA and Vanilla PVT on ModelNet40

Quantitative results demonstrated that the Baseline PVT achieved an overall accuracy (OA) of **93.70%** and a mean class accuracy (mAcc) of 91.45%. DSVA-PVT, configured with $k = 16$ and top-8 salient voxel selection, attained an OA of 93.07% (a decrease of 0.63 points) and an mAcc of 91.01%. When increasing the k-NN proposal size to $k = 32$ and selecting top-12 salient voxels, the OA further decreased to 92.99%, and the mAcc decreased slightly to 90.85%. In terms of inference speed, the Baseline PVT achieved approximately 120 shapes per second, whereas the DSVA-PVT ($k = 16$) performance decreased by approximately 29%, processing around 85 shapes per second.

The slight reduction in accuracy observed with DSVA-PVT suggests that fixed-window attention is sufficiently effective for the structured, clean data in ModelNet40, making dynamic voxel selection less impactful. Moreover, increasing the proposal size (e.g. k=32) further exacerbates performance drops, likely due to the inclusion of less informative voxels that dilute saliency-driven attention.

#### 5.1.1 DSVA and Vanilla PVT on ScanObjectNN

Quantitatively, the Baseline PVT achieved an overall accuracy (OA) of 76.34% and mean class accuracy (mAcc) of 73.10%. DSVA-PVT with k-NN proposals $k = 24$ and selecting the top-10 salient voxels improved performance, achieving an OA of **76.75%** (a 0.41-point gain) and an mAcc of 73.58% (a 0.48-point gain). However, increasing k-NN proposals to $k = 32$ and top-12 voxel selection resulted in re-

duced accuracy, with an OA of 74.01% and mAcc of 71.22%. Per-class performance notably improved for the "Chair" and "Table" classes but slightly decreased for "Monitor."

Inference speed measurements showed that the Baseline PVT processed approximately 60 shapes per second, whereas DSVA-PVT ($k = 24$) processed about 45 shapes per second, representing a 25% slowdown. Qualitatively, DSVA enhanced attention to key structural features, such as thin chair legs and keycaps, and effectively reduced confusion between visually similar classes (e.g., "sofa" and "bed"). Analysis suggested the accuracy gains justified the moderate computational overhead, with potential for speed improvements through custom kernel optimization.

**Effect of Increasing $k$ Across Layers.** In additional ablation experiments, we varied the neighbor count $k$ across DSVA layers, increasing it in deeper blocks (e.g., $k = [8, 12, 16]$) to capture more abstract context. On ModelNet40, this layer-wise increase in $k$ improved performance under the no point-attention (P-Attn) setting, achieving a higher OA of 91.37% compared to the baseline PVT (91.21%). However, on ScanObjectNN, the same strategy led to a degradation in accuracy (OA dropped from 73.04% to 72.76%), likely due to the inclusion of noisy or irrelevant neighbors in cluttered scenes.

This suggests that while deeper aggregation improves learning in clean data, it may overfit to spurious structure in real-world scans. Larger $k$ values implicitly increase the model's receptive field and capacity to aggregate distant context. On clean datasets this is helpful, but on noisy ones it may lead to overfitting spurious patterns, similar to how deeper networks can overfit small datasets without sufficient regularization. This strategy parallels the way receptive fields grow in convolutional neural networks (CNNs), where deeper layers integrate information from a broader spatial extent. Similarly, it reflects the hierarchical neighborhood expansion in PointNet++ [14], where deeper layers aggregate features from increasingly larger regions. By increasing $k$ across DSVA layers, we emulate this multi-scale design, allowing early layers to focus on fine-grained geometric detail and later layers to attend to semantically broader voxel regions.

These results indicate that DSVA adapts to noisy scenarios. However, the performance degradation observed at higher voxel selection parameters (e.g. $k = 32$) suggests potential overfitting to irrelevant noise or less meaningful features. In fact, we observed overfitting on ScanObjectNN was a large problem unless we regularized aggressively with values like $0.3$ for dropout. The modest gains for specific classes like Chair and Table confirm the model's improved discriminative ability for complex objects. Although DSVA introduces a computational overhead of around 25%, this is traded off for accuracy gains. Our implementation is solid but not nearly as efficient as it could be, especially if further optimized through customized CUDA kernels.

| Component | Time Complexity |
|---|---|
| Voxelization & mask | $O(N + V)$ |
| Brute-force KNN | $O\big(M^2\big)$ |
| Edge scoring (MLP) | $O\big(M\,k\,D\big)$ |
| Mask selection (top-$k$) | $O\big(M\,k\big)$ |
| Sparse cross-attention | $O\big(M\,k\,D\big)$ |

Table 3. Per-component time complexity for dynamic sparse voxel attention, where $N$ = number of points, $V$ = total voxels, $M$ = non-empty voxels, $k$ = neighbors per anchor, and $D$ = feature dimension.

### 5.2. Analysis of Time Complexity

Training on ModelNet40 for 200 epochs at 2 min/epoch took roughly 7-8 hours on one high-end GPU (V100/A100), while ScanObjectNN, at a rate of 6 min/epoch, took 20 hours for 200 epochs. Despite optimized CUDA kernels and efficient implementations, ScanObjectNN inherently imposes higher computational and memory demands compared to ModelNet40. This is primarily due to ScanObjectNN's real-world scan characteristics, including cluttered backgrounds, partial occlusions, and irregular point distributions. Such complexities necessitate processing denser, more unstructured data, significantly increasing both computational overhead and memory consumption.

## 6. Conclusion

Our experiments demonstrated that DSVA-PVT improved performance on ScanObjectNN, particularly due to its ability to selectively focus on salient voxels amidst noisy and cluttered data. Conversely, on the cleaner ModelNet40 dataset, baseline PVT outperformed DSVA-PVT, highlighting that dynamic voxel attention provides less benefit when data complexity is lower. Overall, DSVA's effectiveness largely depends on the dataset complexity, with its strengths most pronounced in challenging, real-world conditions.

## 7. Future Work

In deeper layers, voxel features become more abstract and globally informative, so a fixed k-NN at every stage may be suboptimal. One strategy is to let each voxel dynamically choose its number of neighbors: for instance, use a small MLP to predict a per-voxel threshold $\tau_i$ so that only those neighbors with scores $s_{ij} > \tau_i$ are kept. Early layers (where geometry is fine-grained) might use a larger $k$, while later layers (where features encode high-level structure) could safely prune more aggressively. Alternatively, the model could learn a schedule—e.g., $k_1, k_2, \ldots$ for each stage—via

backprop, allowing deeper layers to focus on fewer, but semantically richer, neighbors.

Currently, DSVA's scoring MLP sees the raw coordinate offset $(p_i - p_j)$ concatenated with features. By replacing that with a learned positional embedding—such as Fourier features or spherical harmonics—we can better capture local curvature and non-Euclidean structures. This learned encoding helps the network distinguish subtle geometric relationships (e.g., voxels on opposite sides of a curved surface) that raw offsets alone might not represent as effectively.

## 8 Contributions & Acknowledgements

**V.S.** implemented the core DSVA architecture, its integration, adapted the training and evaluation pipeline for ModelNet40, and developed the confusion-matrix and saliency visualization code.

**M.B.** wrote the ScanObjectNN data-loading and preprocessing modules (including normal estimation), adapted the data pipelines for ScanObjectNN, and assisted with end-to-end validation of inference results.

**Both** authors jointly chose and tuned hyperparameters for DSVA vs. window-attention experiments, ran large-scale training jobs on Colab Pro+, performed literature review and wrote the Related Work section, and managed the experiment scripts (hyperparameter sweeps, checkpointing, and WandB logging).

We leveraged the public implementation from Wan *et al*. [26] as our starting point for the Point–Voxel Transformer backbone, making necessary modifications to support dynamic sparse voxel attention (DSVA) and ScanObjectNN compatibility. The following external libraries and packages were used without modification:

We thank the Stanford Research Computing Center for providing GPU access and containerization support. We also acknowledge the open-source community for maintaining the above libraries and for the original PVT codebase on GitHub. Finally, we thank our course TA, Chaitanya Patel, for his guidance and feedback throughout the project.

## References

[1] G. Bertasius, H. Wang, and L. Torresani. Is space-time attention all you need for video understanding? *arXiv preprint arXiv:2102.05095*, 2021.

[2] Y. Cao, Z. Wu, Y. Zhang, L. Zhang, Y. Zhou, Y. Wan, and Z. Wang. Sparse voxel transformer: Learning 3d shape representation from sparse voxel grids. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[3] N. Caswell. tqdm: A fast, extensible progress bar for python. `https://github.com/tqdm/tqdm`, 2018. Version 4.0.

[4] Z. Chen, Z. Zhou, X. Zhu, and D. Lin. Voxelnext: Next generation voxel-based 3d detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[5] A. Collette. h5py: Hdf5 for python. *Proceedings of the Python in Science Conference*, pages 51–56, 2013.

[6] G. Crosetto. plyfile: Pure python read/write library for stanford ply files. `https://pypi.org/project/plyfile`, 2019. Version 0.7.

[7] C. Guo, Y. Guo, H. Yuan, Q. Hu, Y. Wang, H. You, J. Li, H. Zhao, and T.-J. Mu. Pct: Point cloud transformer. *arXiv preprint arXiv:2012.09688*, 2020.

[8] S. K. Lam, A. Pitrou, and S. Seibert. Numba: A llvm-based python jit compiler. `https://numba.pydata.org`, 2015. Version 0.48.

[9] lanpa. TensorBoardX: Tensorboard support for pytorch. `https://github.com/lanpa/tensorboardX`, 2017. Version 1.2.

[10] L. Liu, P. He, and W. Chen. Transformer acceleration with dynamic sparse attention. *arXiv preprint arXiv:2110.11299*, 2021.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. *NeurIPS Deep Learning Workshop*, 2019.

[12] B. Peterson. Six: Python 2 and 3 compatibility library. *Python Package Index*, 2009. Version 1.15.0, `https://pypi.org/project/six`.

[13] Python Software Foundation. Python: A dynamic, high-level programming language. `https://www.python.org`, 2023. Version 3.7.

[14] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5099–5108, 2017.

[15] P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018.

[16] P. Sun, W. Zeng, Y. Zhang, Y. Chen, W. Hua, T. He, R. Liao, and R. Urtasun. Voxel transformer for 3d object detection. In *Advances in Neural Information Processing Systems*, 2021.

[17] M. A. Uy, Q.-H. Pham, B.-S. Hua, D. T. Nguyen, and S.-K. Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *International Conference on Computer Vision (ICCV)*, 2019.

[18] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. NumPy: The fundamental package for scientific computing with python. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[19] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. SciPy1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020.

[21] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[22] Weights & Biases, Inc. Weights & Biases. `https://wandb.ai`, 2018. Version 0.10.

[23] R. Wightman. timm: Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2021. Version 0.6.

[24] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. Oral presentation; Project page at 3D Deep Learning.

[25] C. Zhang, Y. Feng, Q. Hu, W. Wu, Z. Hu, Z. Yan, X. Zhao, and Y. Guo. Pvt: Point-voxel transformer for point cloud learning. *arXiv preprint arXiv:2108.06076*, 2022.

[26] C. Zhang, H. Wan, S. Liu, X. Shen, and Z. Wu. PVT: Point–Voxel Transformer for 3D Deep Learning. `https://github.com/HaochengWan/PVT`, 2021. GitHub repository.

[27] H. Zhao, L. Jiang, J. Jia, P. H. S. Torr, and V. Koltun. Point transformer. *arXiv preprint arXiv:2012.09164*, 2020.

[28] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2020.