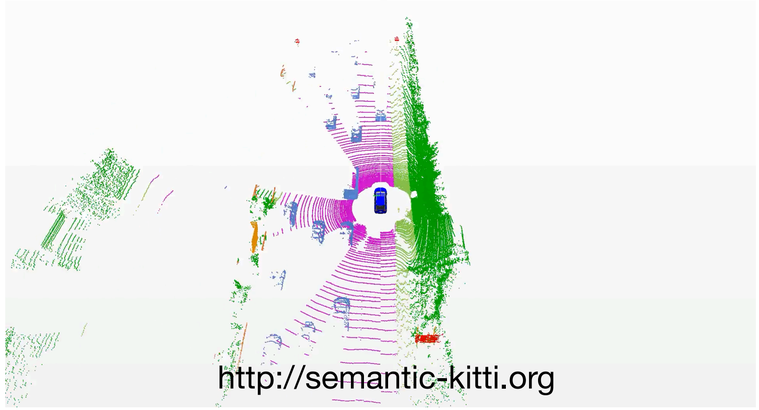
MIT HAN LAB

Lecture 18
Efficient Point Cloud Recognition

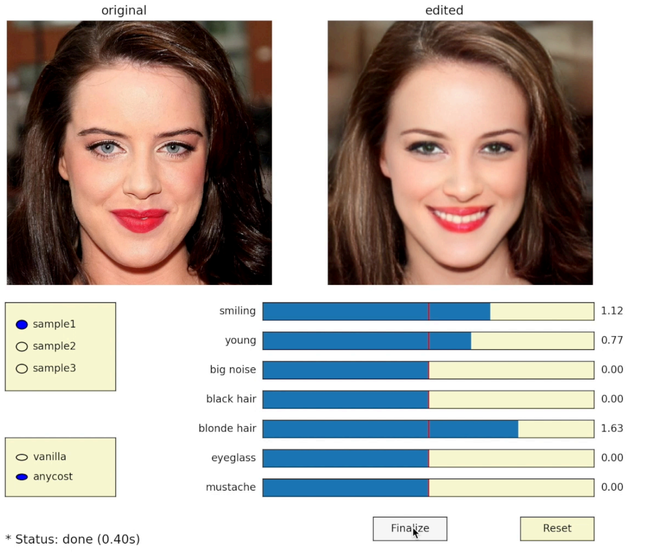
Song Han
songhan@mit.edu

Application-Specific Optimizations for Efficient AI Computing

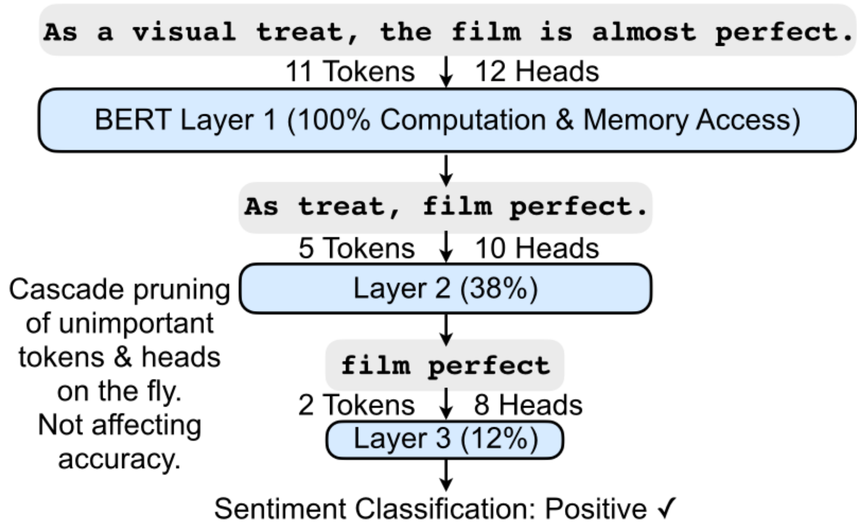


Point cloud: 3D spatial redundancy (this lecture)

Videos: temporal redundancy



GANs: 2D spatial redundancy

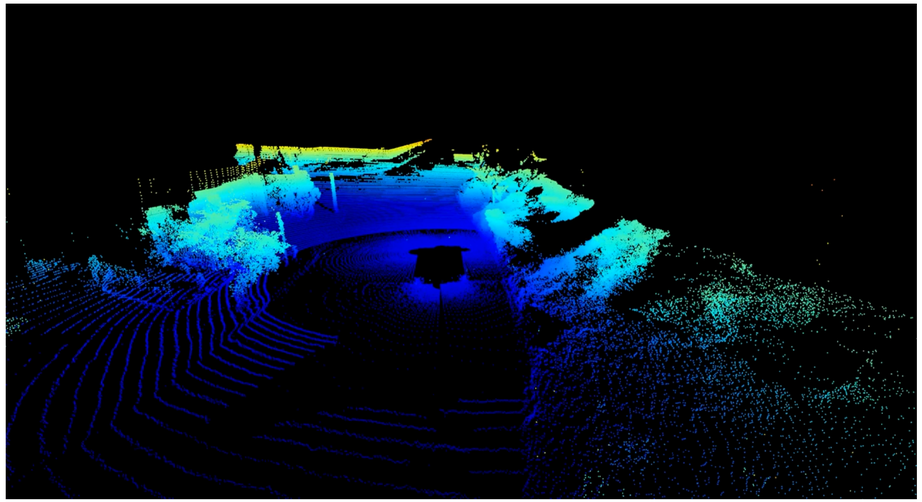


NLP: token redundancy

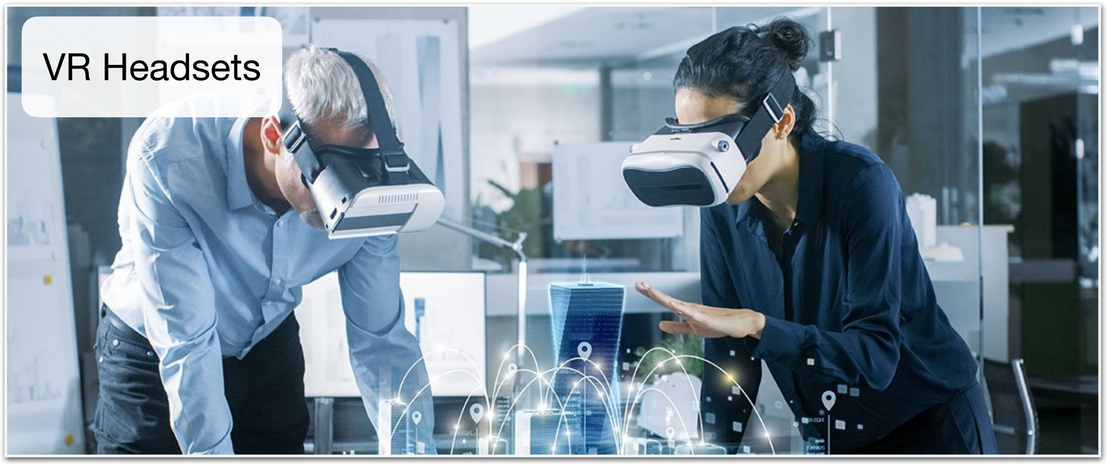
Lecture Plan

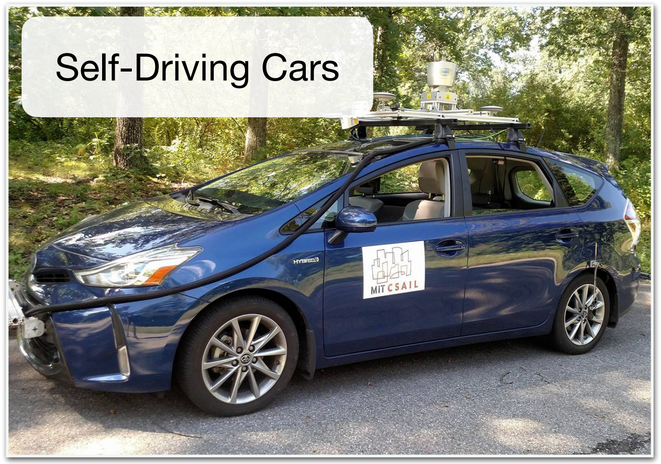
Today we will cover:

1. 3D sensor and point clouds;
2. Different ways of representing and processing 3D data;
3. Efficient algorithms for deep learning on point clouds;
4. Efficient system and hardware support for deep learning on point clouds.



3D Sensors


VR Headsets


Self-Driving Cars
MIT CSAIL

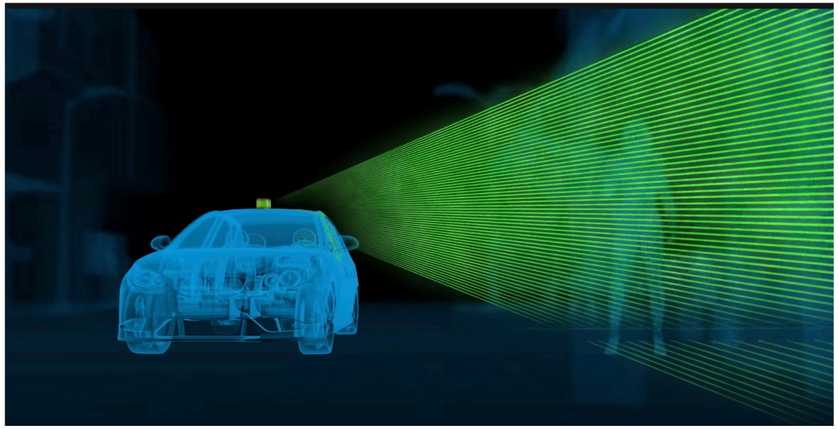

iPhones & iPads
LiDAR Scanner

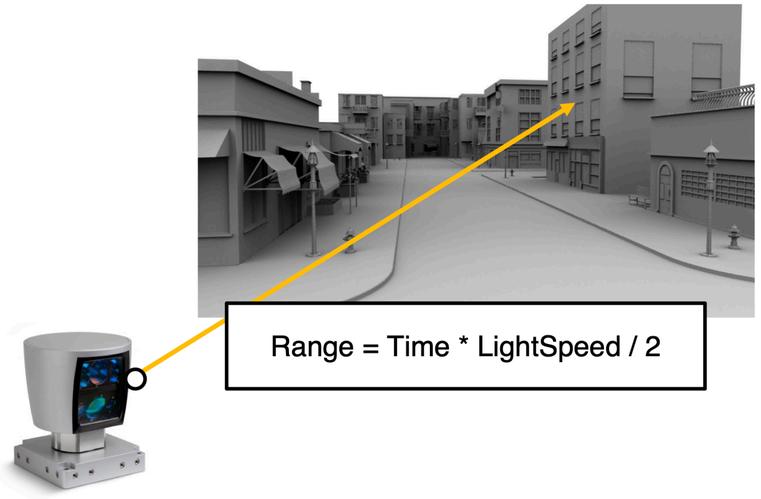

Drones

3D Sensors

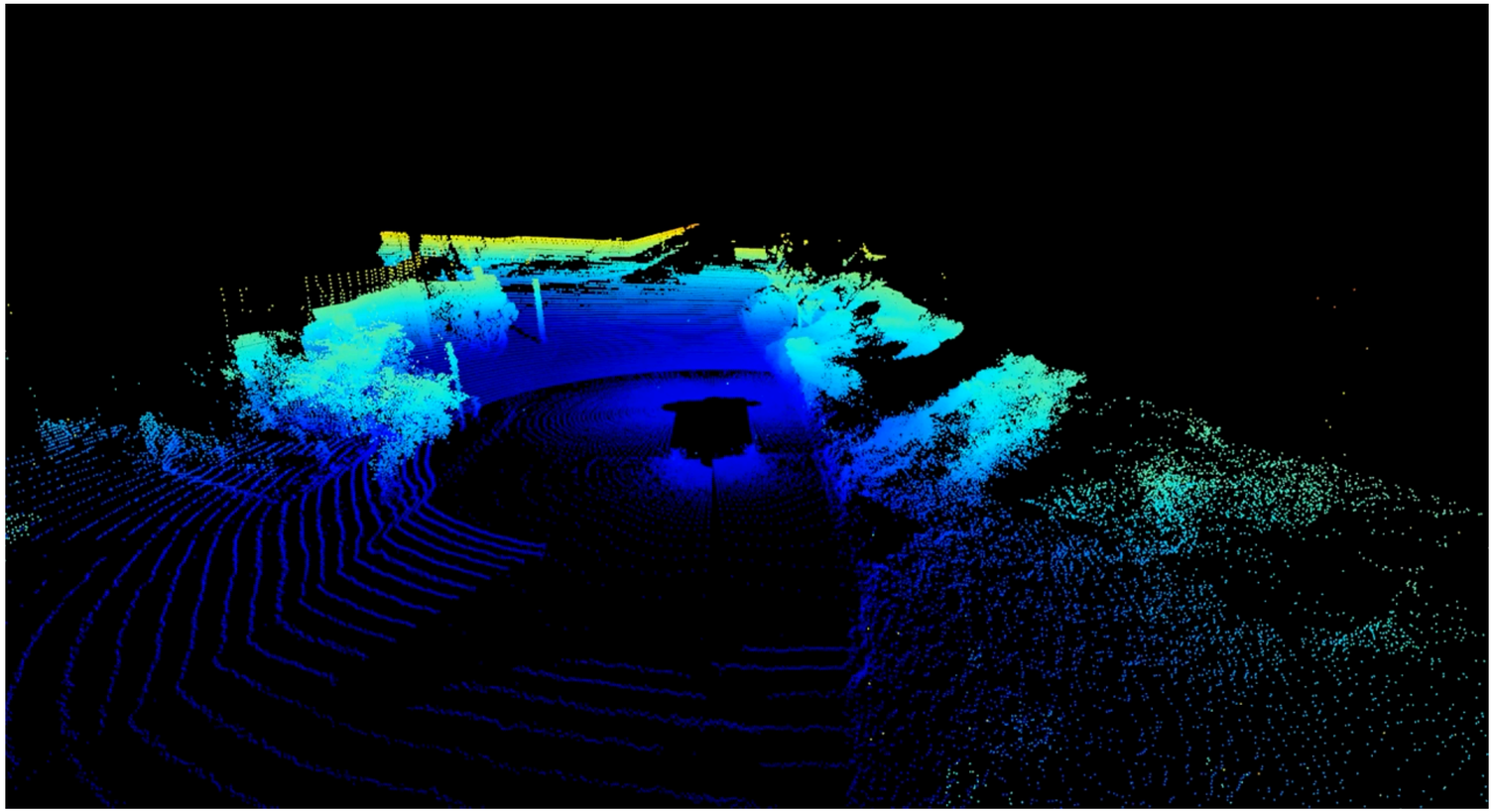
LiDAR

LiDAR Sensor





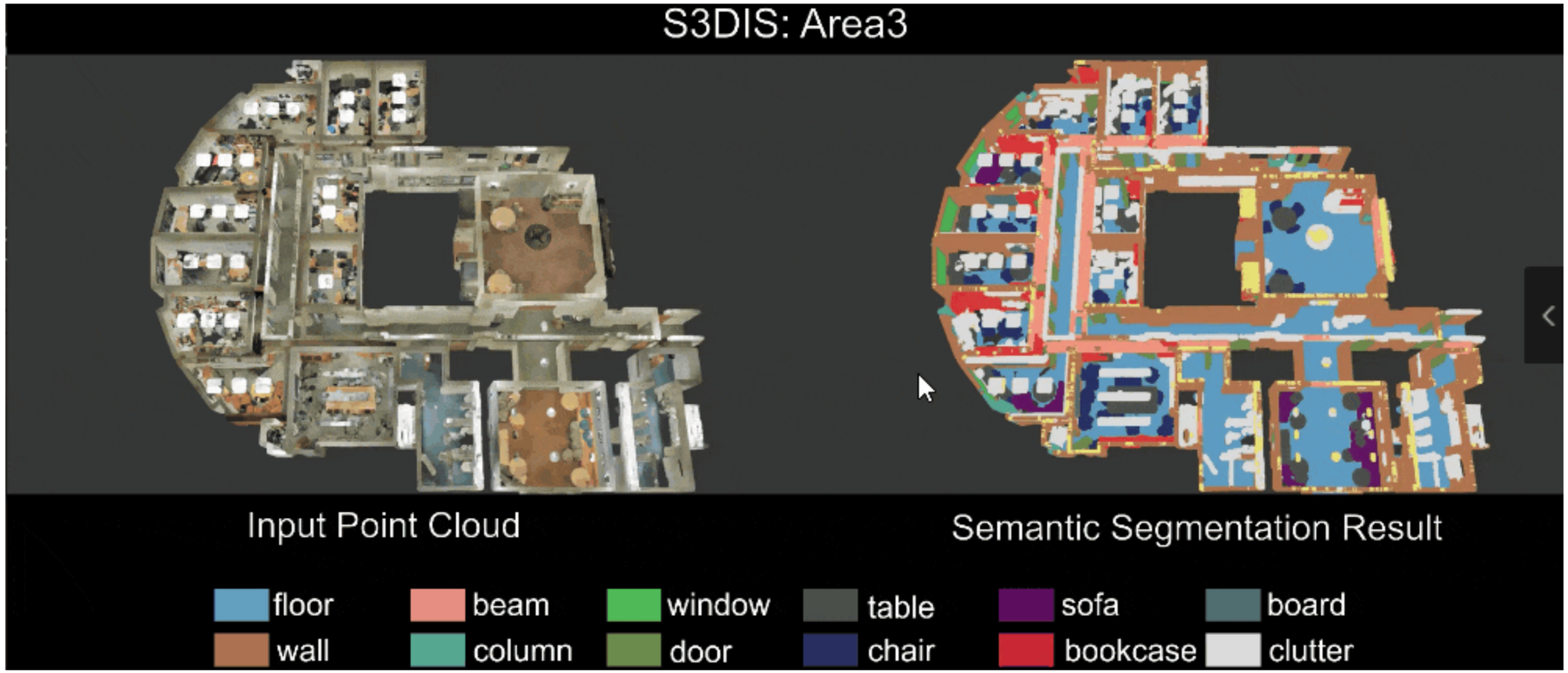
Data: 3D Point Clouds



A point cloud $P = \{(p, f)\}$ is a collection of 3D points $p = [x, y, z]$, each with C dimensional feature $f \in \mathbb{R}^C$.

Applications

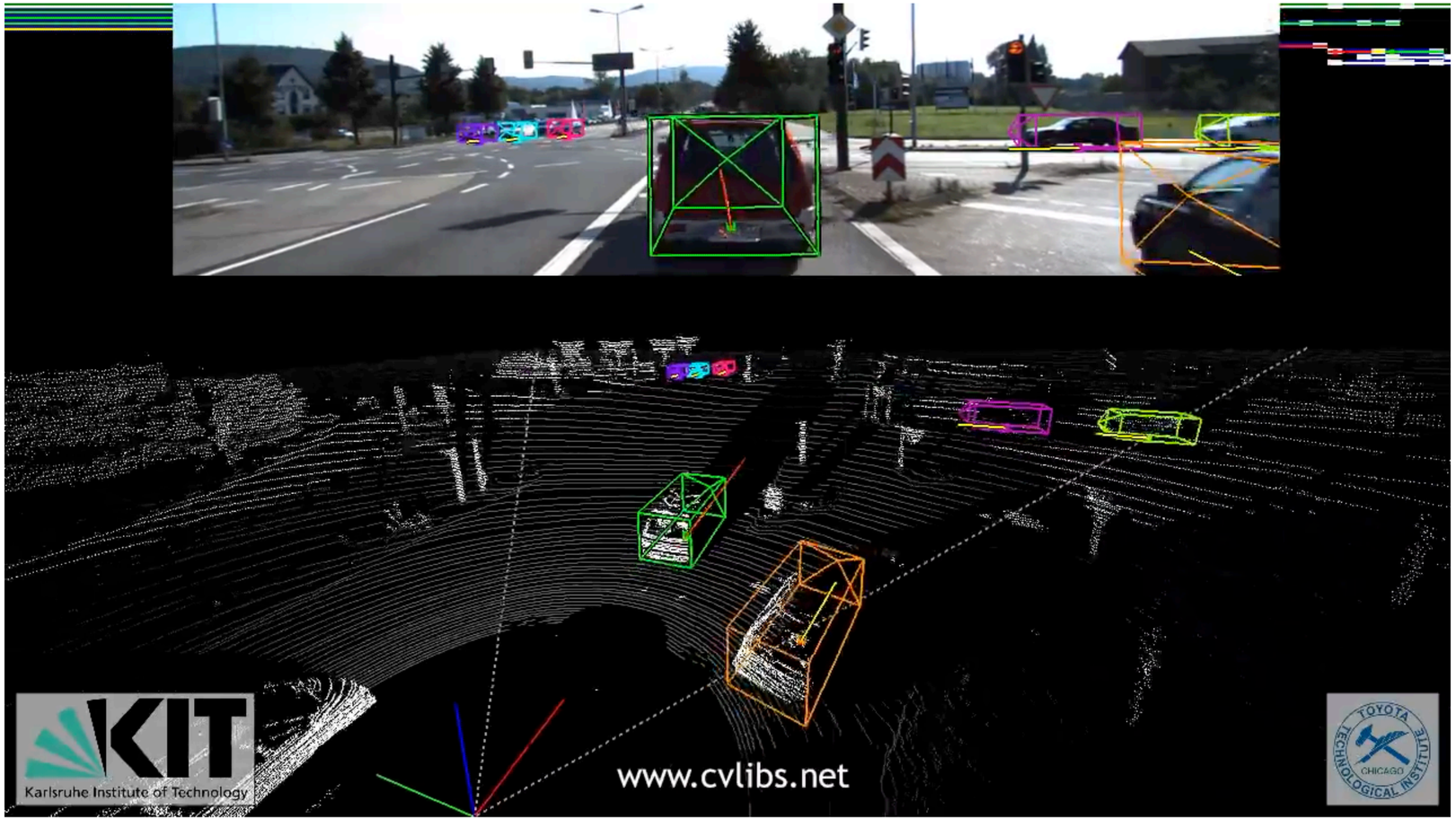
Augmented Reality (AR)



3D Semantic Parsing of Large-Scale Indoor Spaces [Armeni *et al.*, CVPR 2016]
RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds [Hu *et al.*, CVPR 2020]

Applications

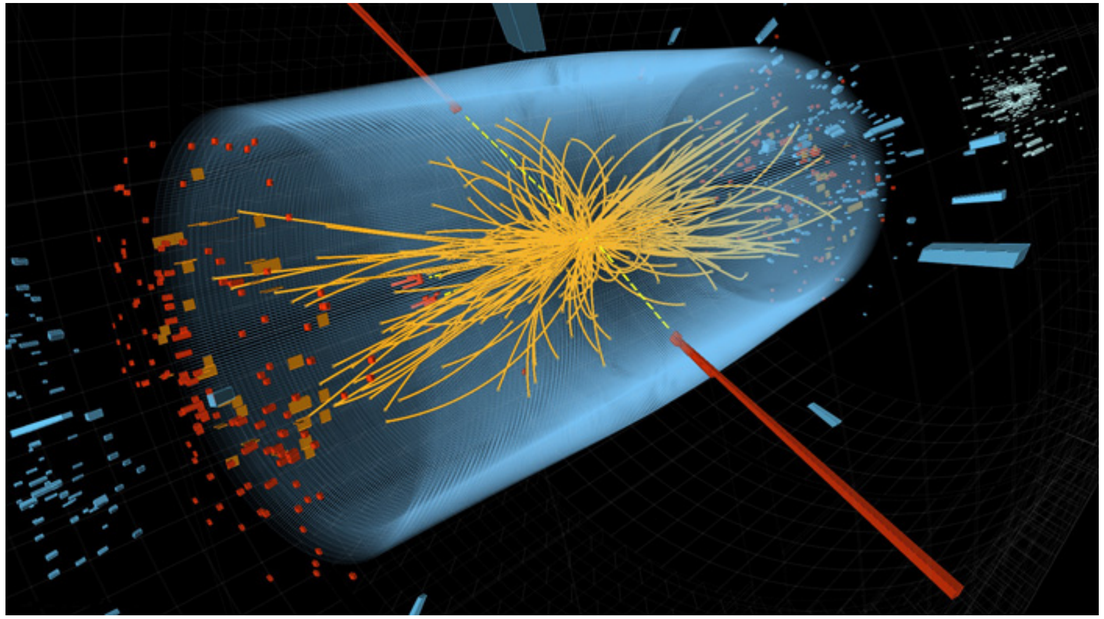
Autonomous Driving



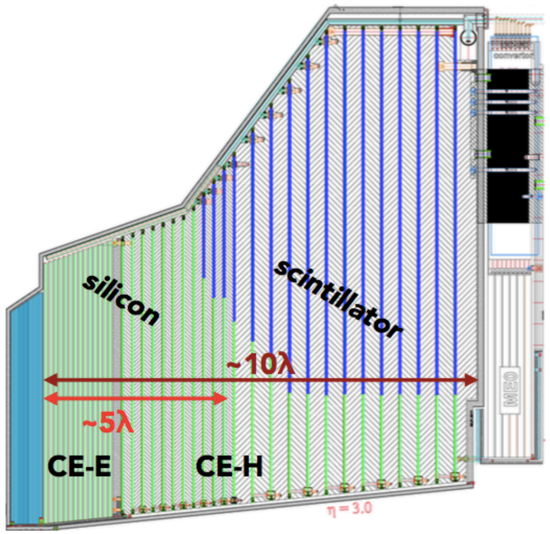
Vision meets robotics: The KITTI dataset [Geiger *et al.*, IJRR 2013]

Applications

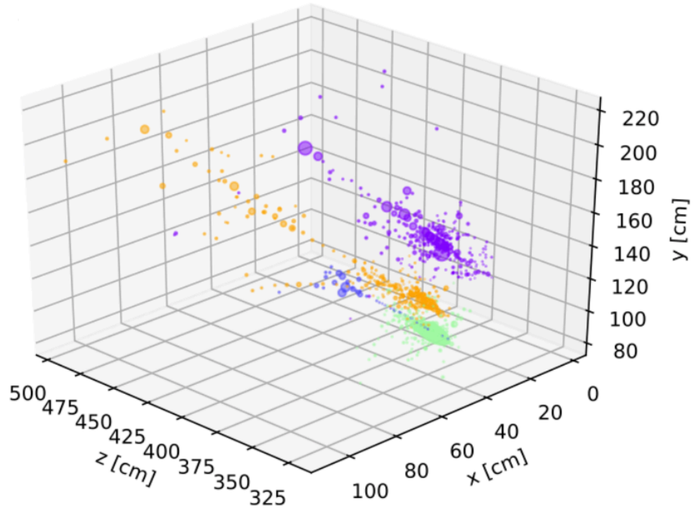
Particle Discovery



Application: High-Energy Particle Physics



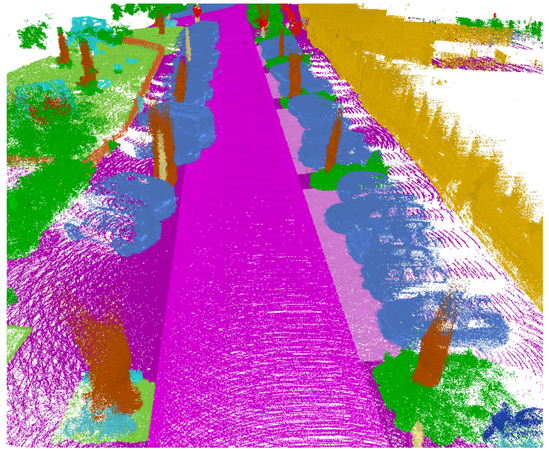
High-Granularity Calorimeter (HGCAL)



Particle Segmentation

Input: (x, y, z, E, t)
Output: particle types

analogy

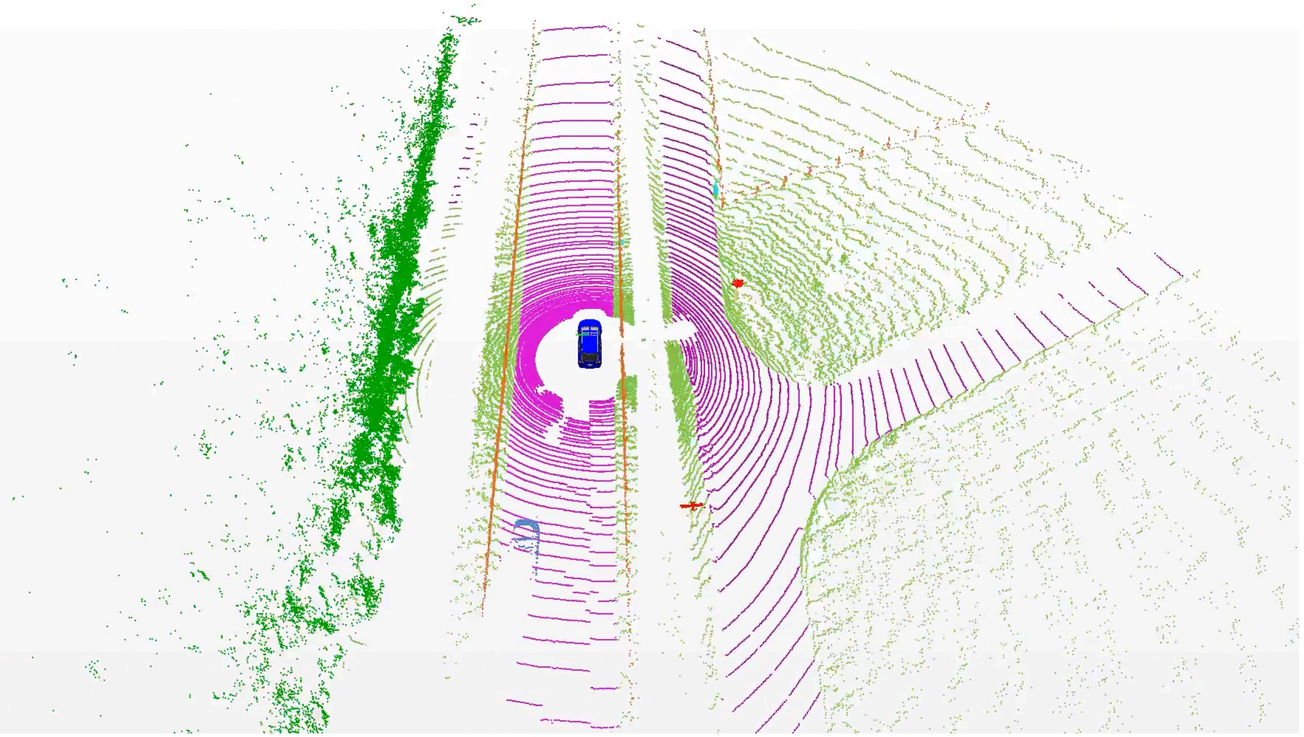


Scene Segmentation

Input: (x, y, z, I)
Output: classes

Challenges

Sparsity and Irregularity



Images:

- Dense and regular
- Can be processed by conventional CNNs

Point clouds:

- Extremely sparse (sometimes <0.1% density)
- Irregularly stored in memory
- Usually processed by specialized operators and systems

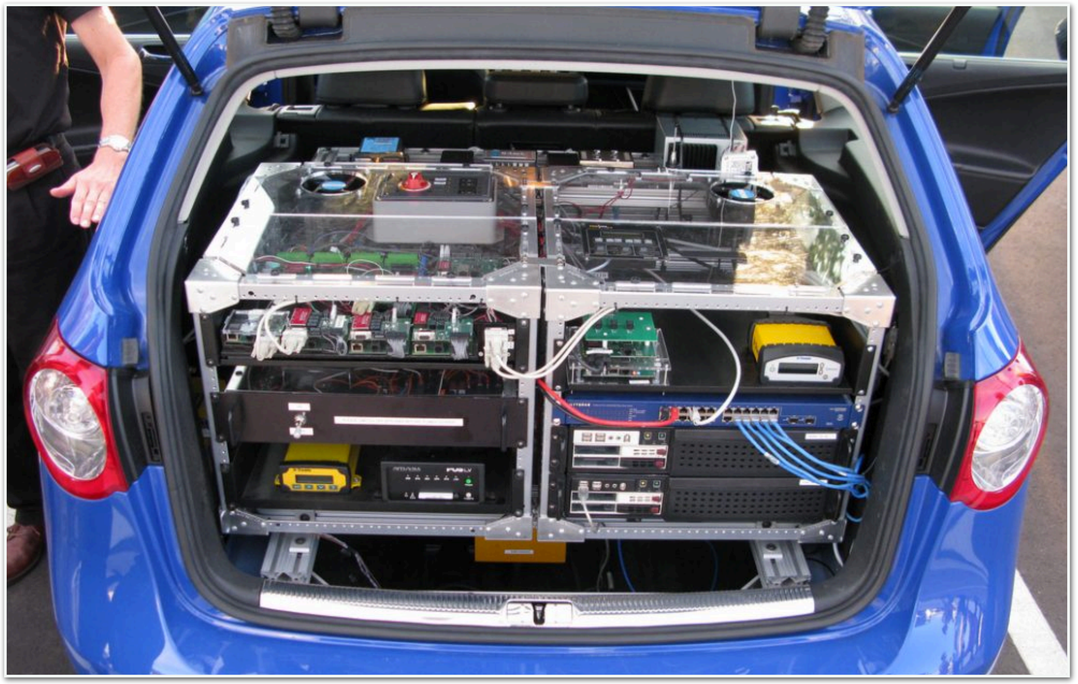
Cat image from the ImageNet dataset and point cloud sample from the SemanticKITTI dataset.

Challenges

Limited Computational Resources

VR/AR Headsets

A full backpack of computers!



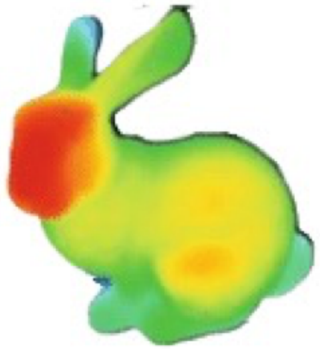
Self-Driving Cars

A whole trunk of workstations!

However, these point cloud deep understanding models must be deployed on resource-constrained edge devices.

Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang *et al.*, ECCV 2020]

Representing 3D Data

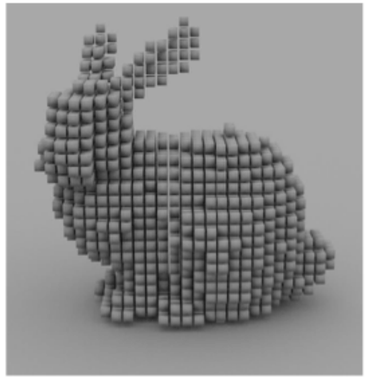


1.1 Multi-view images

Projection

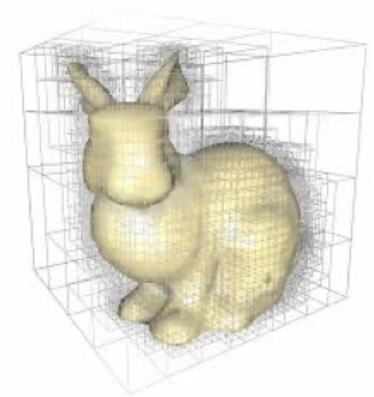
1.3 Point Cloud

Voxelize



1.2 Voxels, 1.4 Sparse Voxels

Voxelize



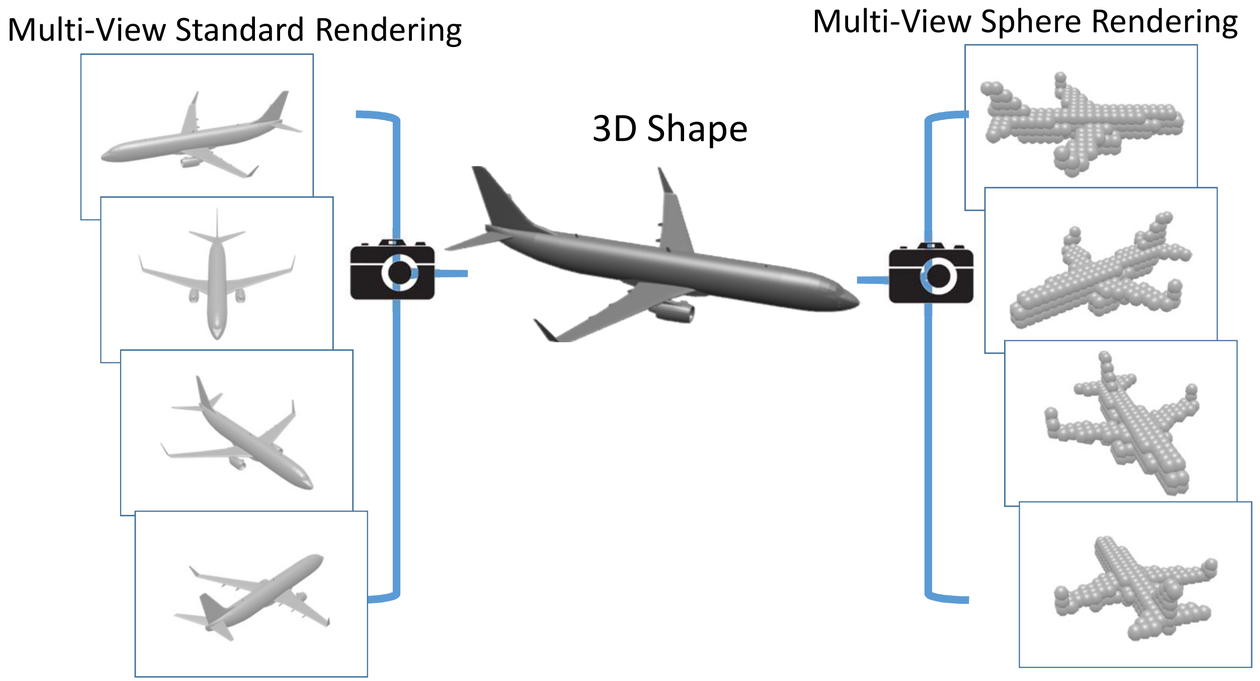
1.5 Octree

1.1 Pseudo / Multi-View Images

Project 3D Point Cloud into Image-Like Representation

Multi-View Images

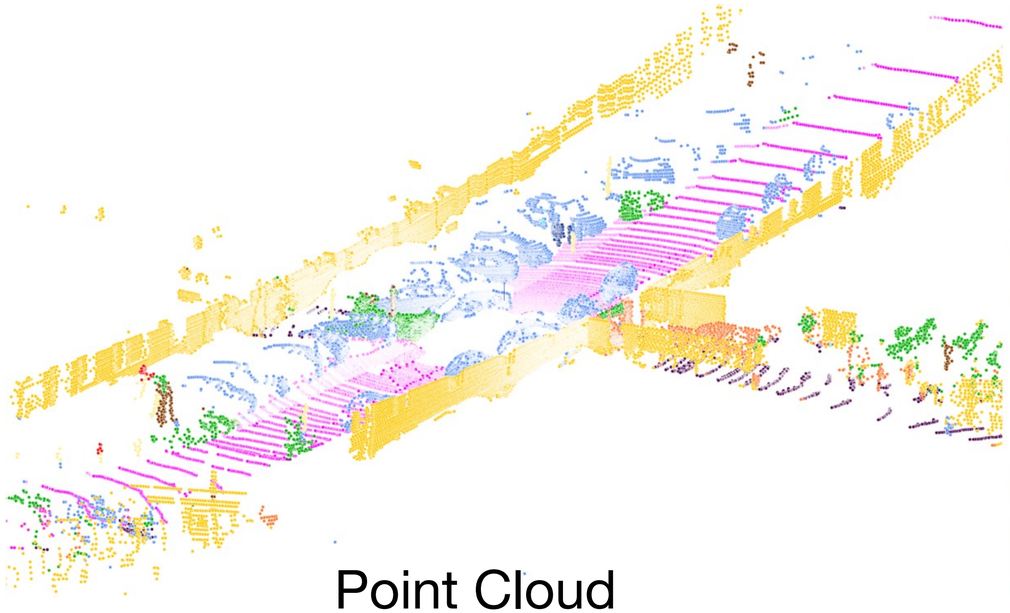
Converting point cloud understanding to image understanding



- Multi-view images can be obtained by rendering the point cloud from different viewing angles.
- One can apply standard 2D CNN on each rendered image and aggregate the results with average / max pooling to obtain the final predictions.

Volumetric and Multi-View CNNs for Object Classification on 3D Data [Qi *et al.*, CVPR 2016]

Range Images



Point Cloud

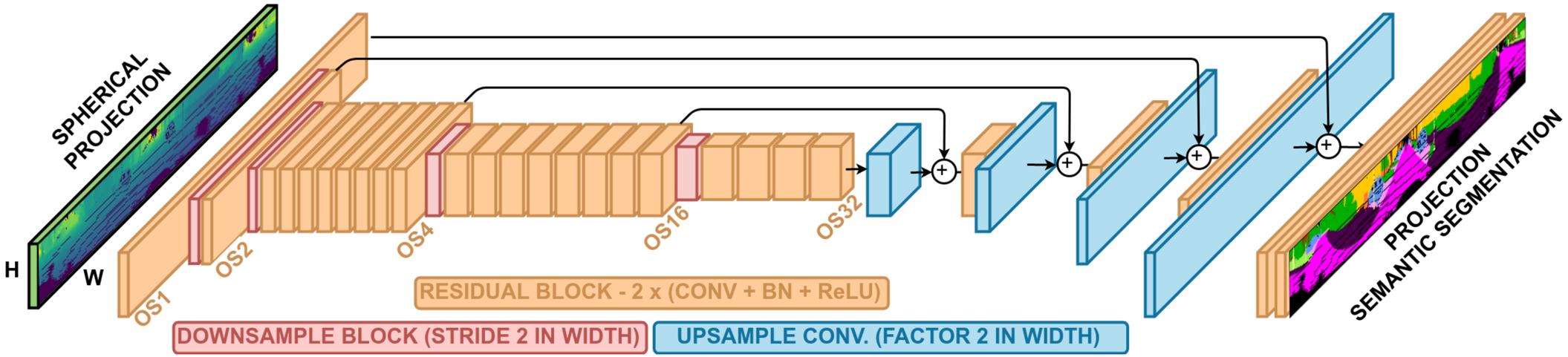
Spherical Projection (Range Images)

f_{down}, f_{up} are the upper/lower bound of vertical field of view of the LiDAR sensor

SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud [Wu *et al.*, ICRA 2018]
SalsaNext: Fast, Uncertainty-aware Semantic Segmentation of LiDAR Point Clouds for Autonomous Driving [Cortinhal *et al.*, arXiv 2020]

Range Images

RangeNet: 3D semantic segmentation on range images

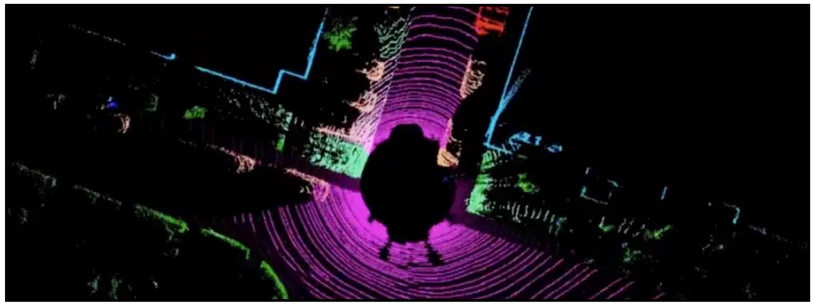


- Similar idea to 2D image segmentation: using a U-net architecture to perform semantic segmentation on spherical projections (range images).
- **Pros:** Can directly utilize 2D CNNs, easy to deploy on hardware;
- **Cons:** Can introduce geometric distortion, works better for semantic segmentation and requires efforts to make it work on detection (**RangeDet**, ICCV'21).

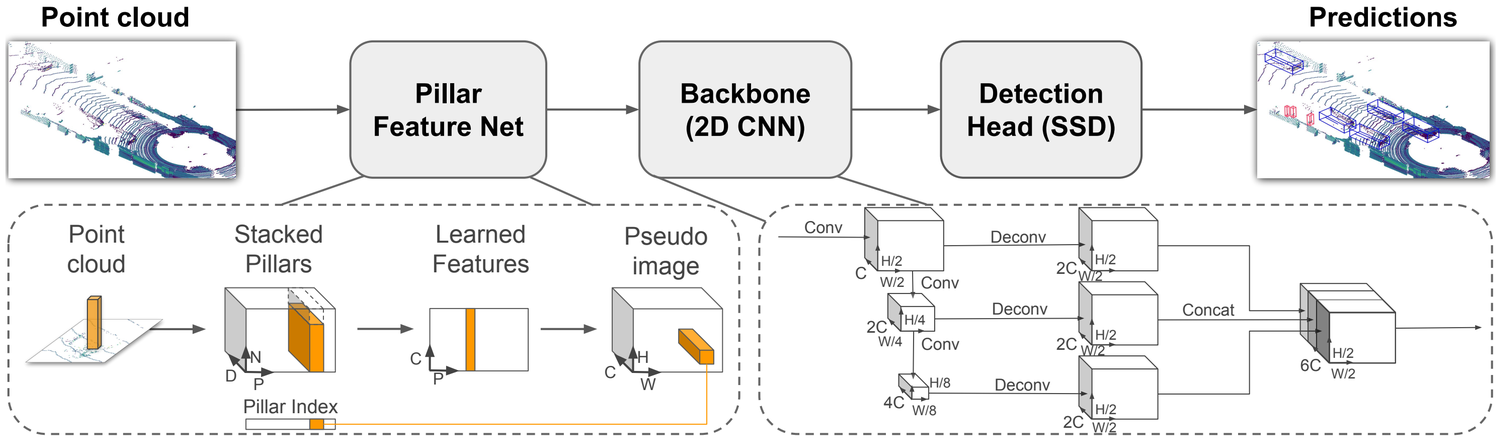
RangeNet++: Fast and Accurate LiDAR Semantic Segmentation [Milioto *et al.*, IROS 2019]

Bird's Eye View (BEV) Projection

PointPillars: Cartesian projection



3D scene in the BEV

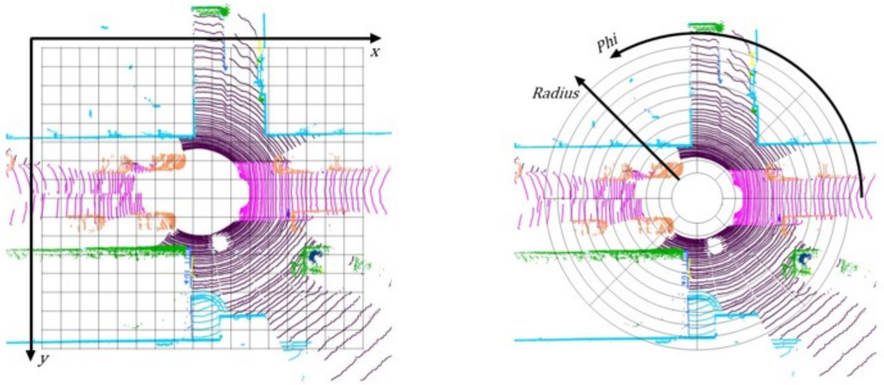


PointPillars flattens the point cloud in the BEV space and applies PointNet within each grid on the BEV plane.
After that, the entire feature map is fed through a 2D CNN to produce object detection predictions.
P: number of pillars (~10000), N: maximum points of points within each pillar (20), D/C: input/output channels (10~64).

PointPillars: Fast Encoders for Object Detection from Point Clouds [Lang *et al.*, CVPR 2019]

Bird's Eye View (BEV) Projection

PolarNet: Polar projection



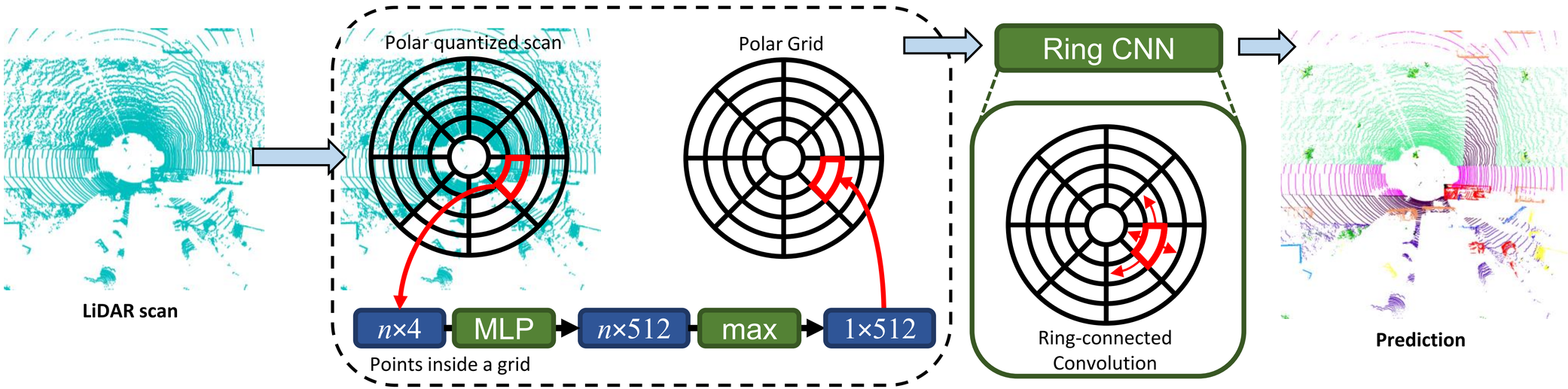
(a) Cartesian BEV (b) Polar BEV

- From Cartesian coordinates to polar coordinates on the xy (BEV) plane:
- $\rho = \sqrt{x^2 + y^2}$,
- $\theta = \arctan \frac{y}{x}$.
- Polar grids are in higher resolution at the origin compared with Cartesian BEV. As a result, usually polar BEV methods require smaller feature map sizes to achieve the same level of accuracy compared with Cartesian BEV methods.

PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation [Zhou *et al.*, CVPR 2020]

Bird's Eye View (BEV) Projection

PolarNet: Polar projection



- Ring convolution in PolarNet: convolution is performed on the ring neighborhood. Notice that $(r,0)$ and $(r,1.99\pi)$ are very close to each other and should be convolved together.

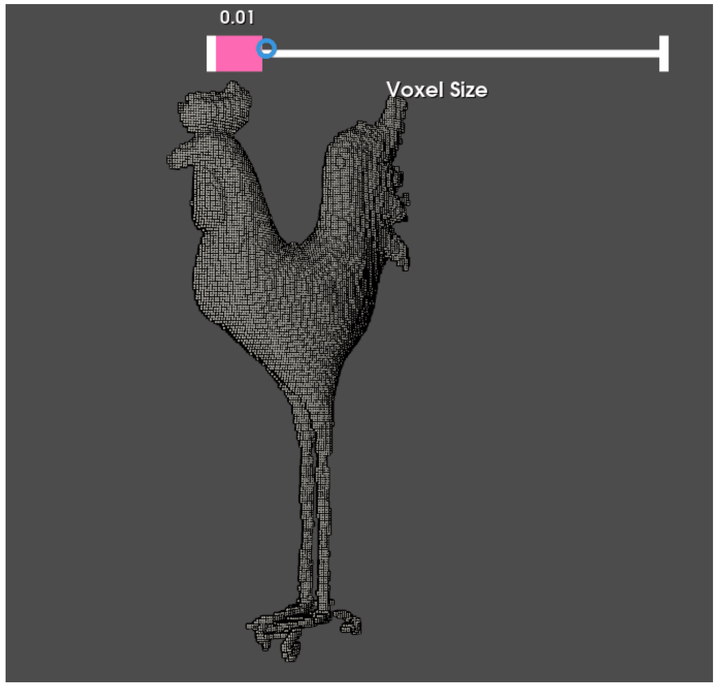
PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation [Zhou *et al.*, CVPR 2020]

1.2 Dense Voxels

Rasterize 3D Point Cloud into Dense Voxel Representation

Voxelization

Converting point clouds to voxels

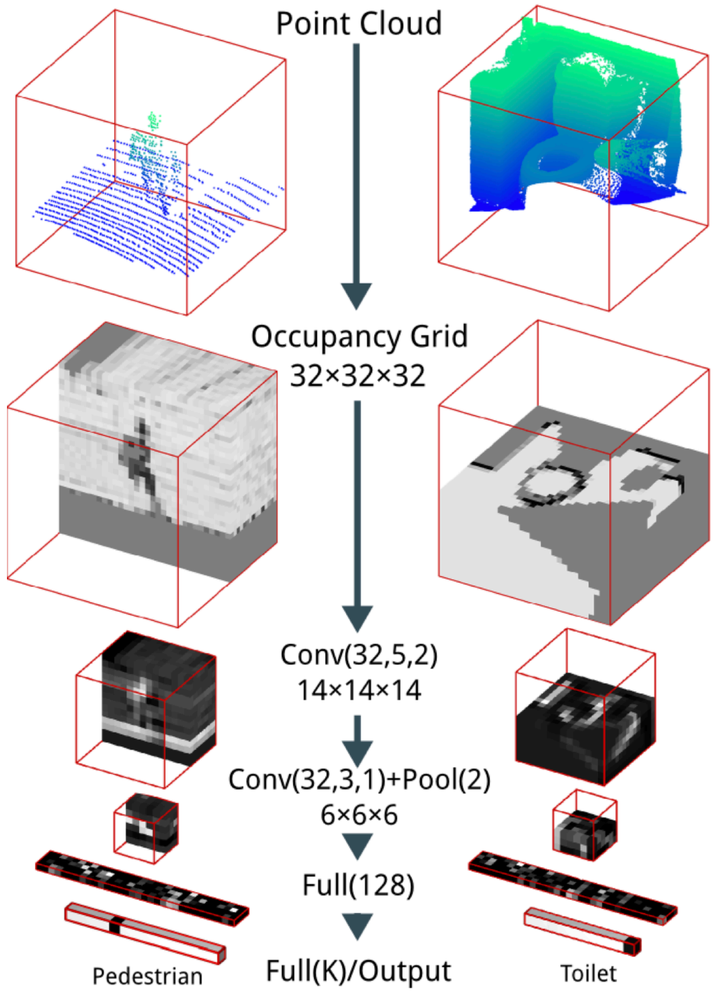


- For a 3D point $p = [x, y, z]$ with features $f \in \mathbb{R}^C$, voxelization under voxel size r means that the coordinates of point p will be quantized to $\hat{p} = [\text{round}(\frac{x}{r}), \text{round}(\frac{y}{r}), \text{round}(\frac{z}{r})]$.
- We create a 4D tensor V of size $H \times W \times D \times C$, where $[H, W, D]$ is the upper bound of $\hat{p}$ for all $p \in P = \{(p_i, f_i)\}$. Then, we insert the feature f_i to location $\hat{p}_i$ of the 4D tensor V.
- Note: We assume the lower bound of $\hat{p}$ is [0, 0, 0] for simplicity.

How to Voxelize Meshes and Point Clouds in Python [Nikolov, 2022]

Deep Learning on Voxels

3D CNN on Voxelized Point Cloud

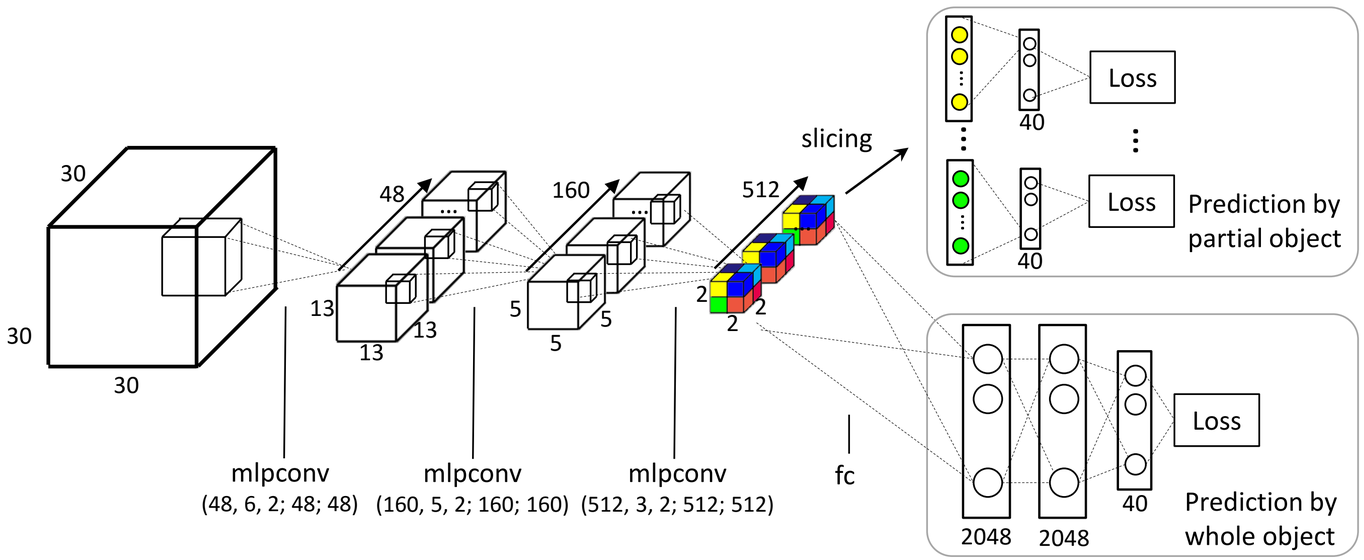


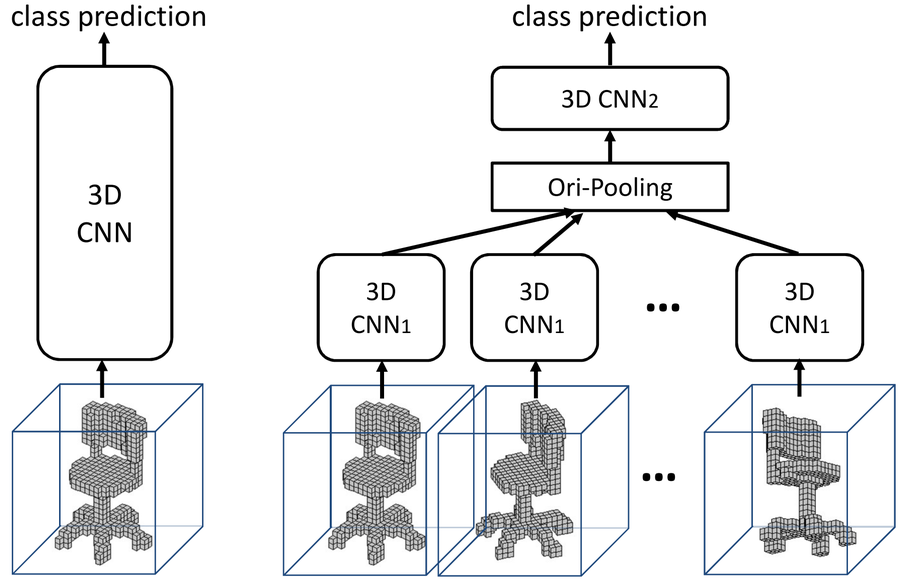
- VoxNet is the early attempt on deep learning on voxel grids.
- The idea is very straightforward: we simply need to generalize 2D CNNs to 3D.

VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition [Maturana and Scherer, IROS 2015]

Deep Learning on Voxels

3D CNN on Voxelized Point Cloud



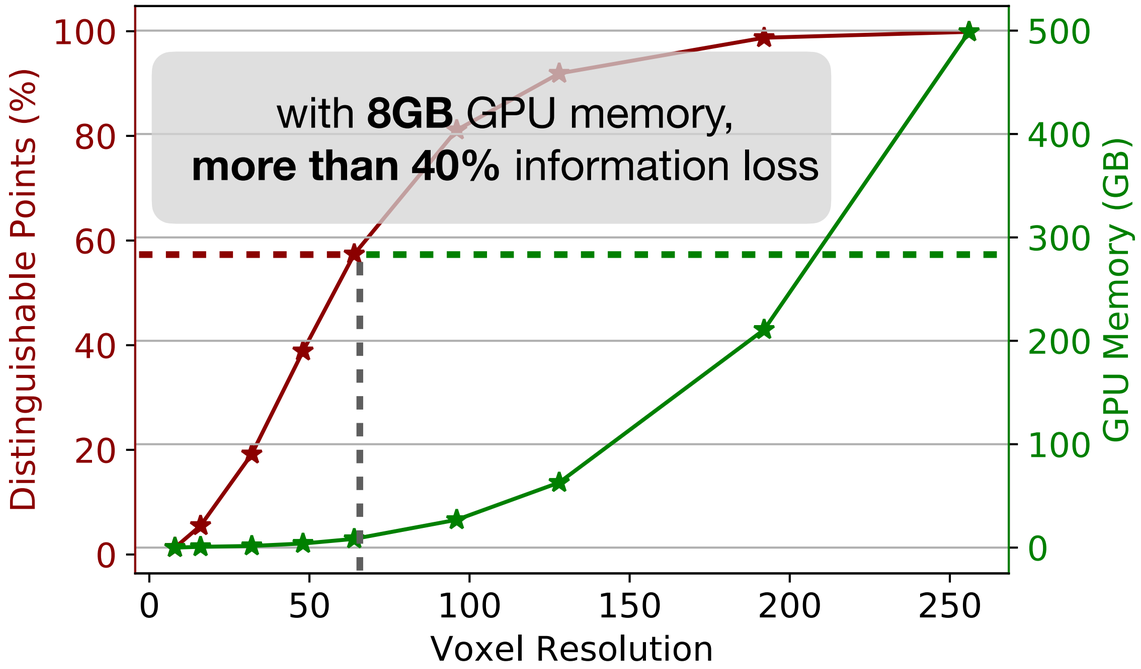


- Subvolume supervision (left): predicting the classification labels from partial voxel feature maps. This can help reduce overfitting.
- Orientation pooling (right): CNNs are not rotationally-invariant. Feeding voxels from different angles to the network and aggregate the results like in MVCNNs.

Volumetric and Multi-View CNNs for Object Classification on 3D Data [Qi *et al.*, CVPR 2016]

Bottlenecks

Cubically-Growing Memory Consumption



Low resolutions lead to **significant information loss**; high resolutions lead to **large GPU memory consumption**.

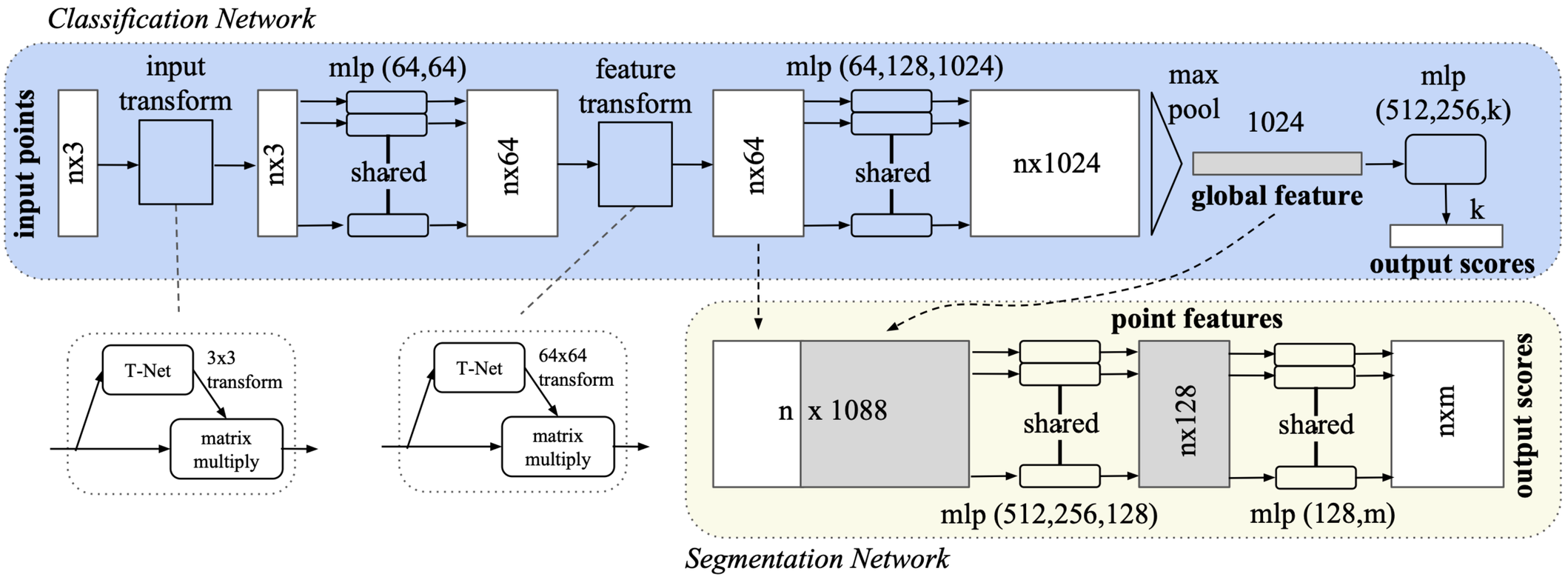
Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

1.3 Point Set

Process 3D Point Cloud as Unordered Set of Points

PointNet

Directly Processing Point Cloud

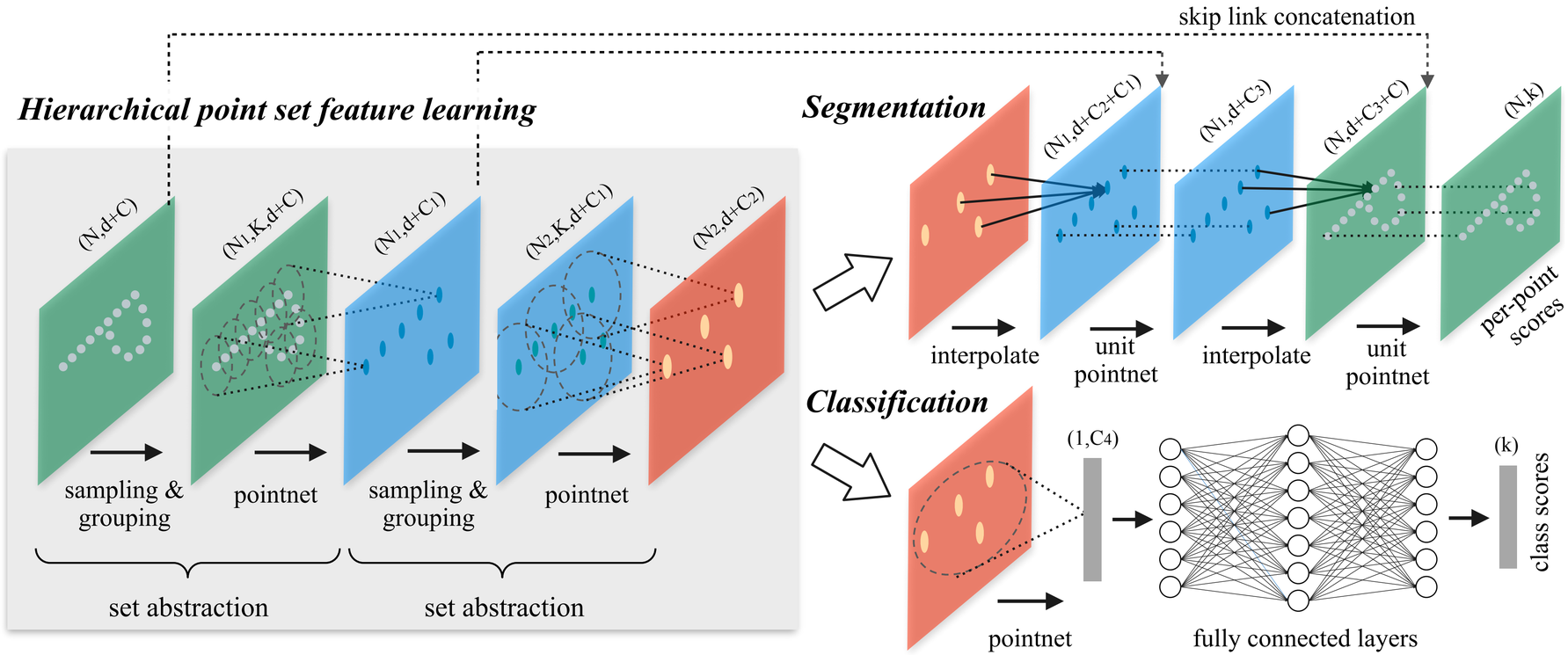


- PointNet models point clouds with MLP layers and a symmetric (i.e. permutation-invariant) function (such as max pooling).
- T-Net: a smaller PointNet learns a linear transformation matrix. For input, the intuition is to rotate the input point cloud to a canonical view.

PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation [Qi *et al.*, CVPR 2017]

PointNet++

Hierarchical PointNet

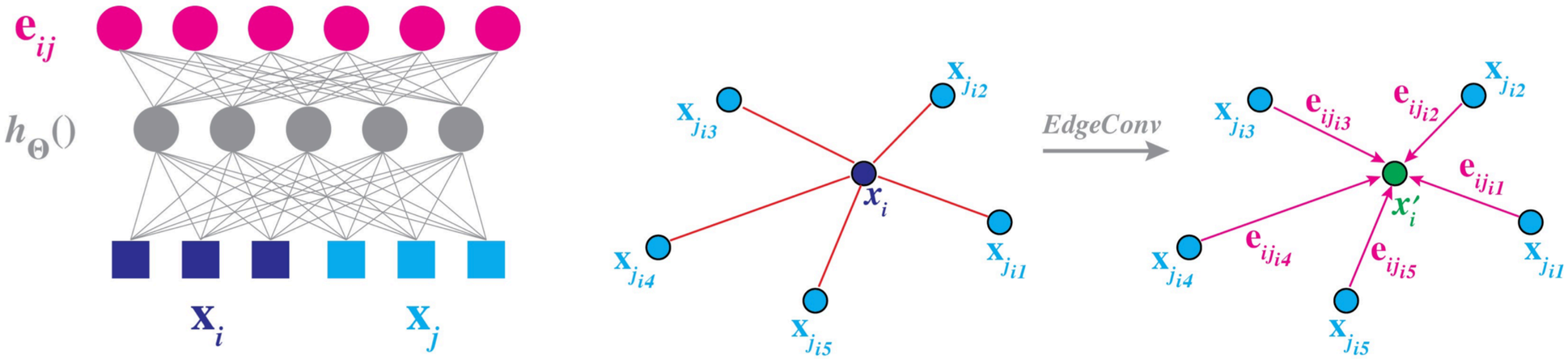


- PointNet++ is a hierarchical version of PointNet.
- It uses furthest point sampling to downsample the point clouds and kNN (fixed number of neighbors) / ball query (fixed distance) to define the neighborhood of each point. PointNet is applied within the neighborhood of all the points.

PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space [Qi *et al.*, NeurIPS 2017]

DGCNN

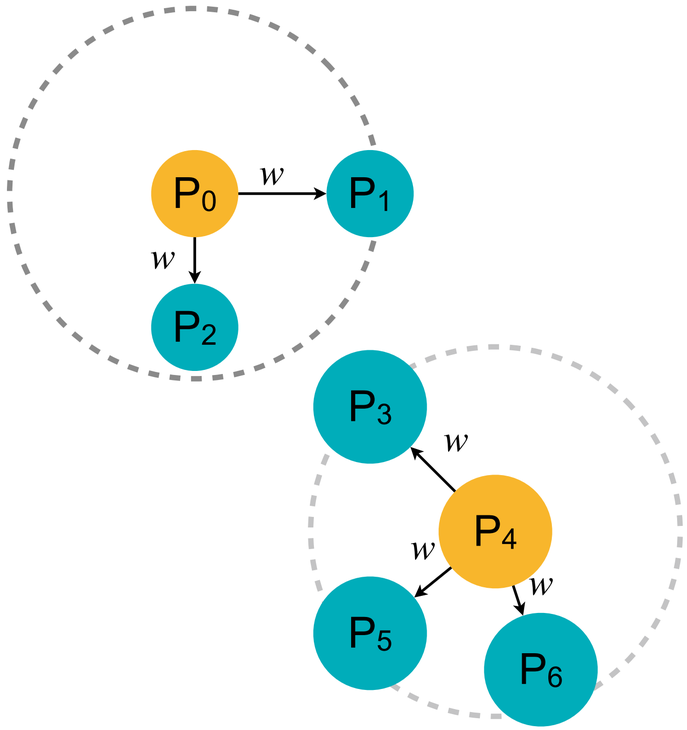
Understanding point cloud as a graph



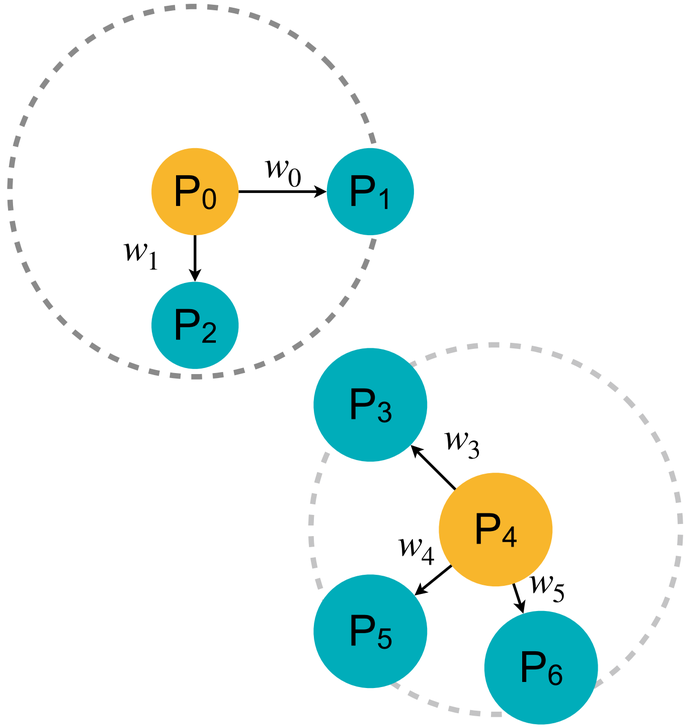
- Different from PointNet++ which defines the neighborhood of each point in the **Euclidean** space, DGCNN defines the neighborhood in the **feature** space, and builds the neighbor graph dynamically.
- DGCNN does not downsample the input.

Dynamic Graph CNN for Learning on Point Clouds [Wang *et al.*, SIGGRAPH 2019]

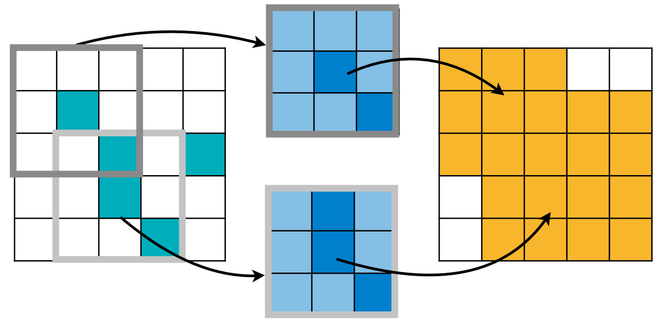
Discussion on PointNet++ and DGCNN



PointNet++/DGCNN:
the same weight for different neighbors



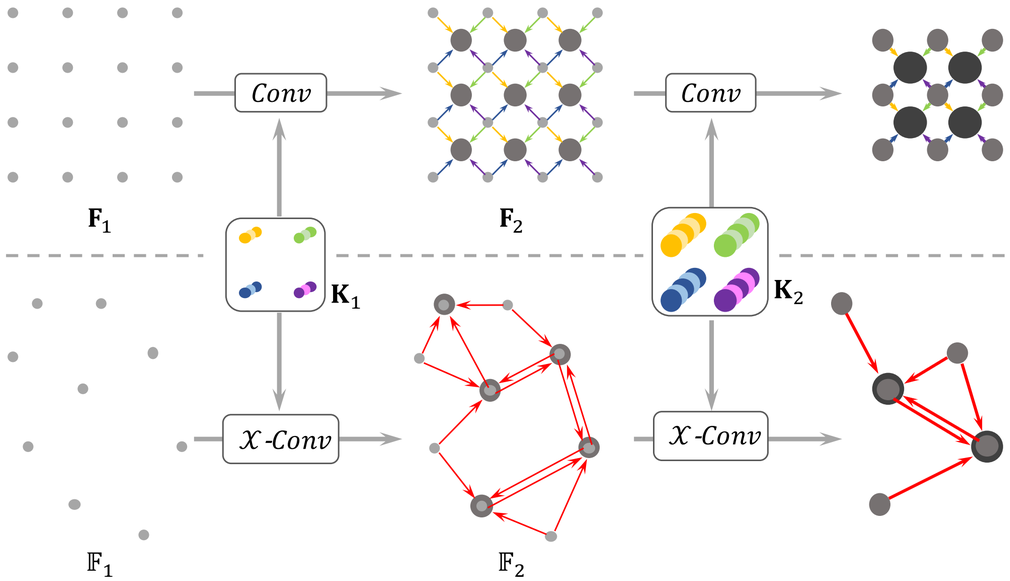
Is it possible to have
different weight for different neighbors?



Analogous to CNNs

PointCNN

Learning to permute the neighborhood points and apply a fixed kernel



ALGORITHM 1: $\mathcal{X}$-Conv Operator

Input : $\mathbf{K}$, p, $\mathbf{P}$, $\mathbf{F}$
Output : $\mathbf{F}_p$ ▷ Features "projected", or "aggregated", into representative point p

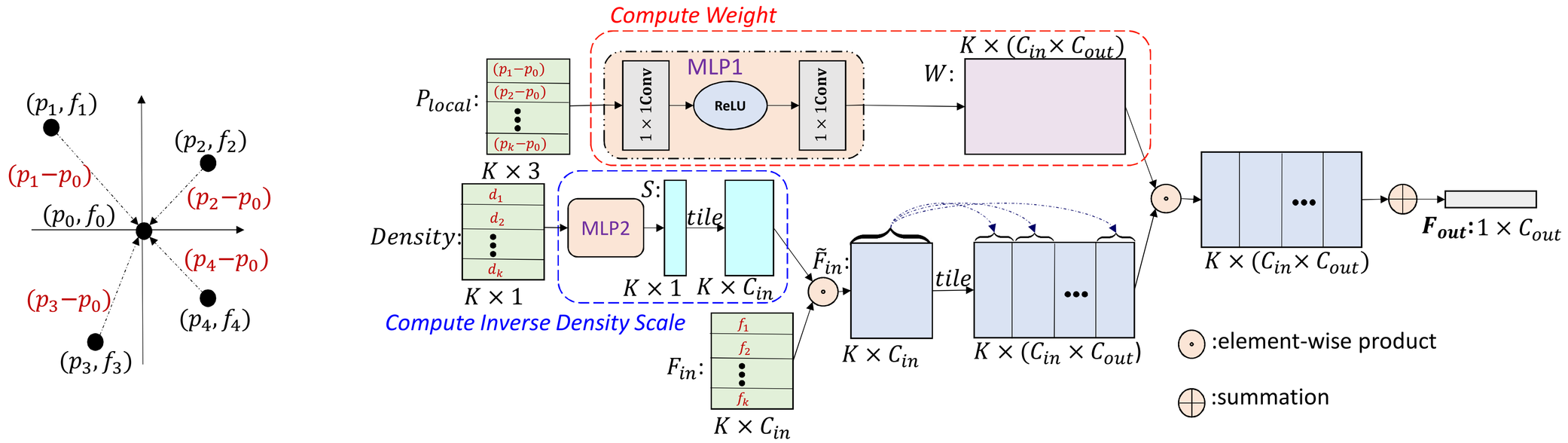
1: $\mathbf{P}' \leftarrow \mathbf{P} - p$ ▷ Move $\mathbf{P}$ to local coordinate system of p
2: $\mathbf{F}_\delta \leftarrow MLP_\delta(\mathbf{P}')$ ▷ **Individually** lift each point into C_δ dimensional space
3: $\mathbf{F}_* \leftarrow [\mathbf{F}_\delta, \mathbf{F}]$ ▷ Concatenate $\mathbf{F}_\delta$ and $\mathbf{F}$, $\mathbf{F}_*$ is a $K \times (C_\delta + C_1)$ matrix
4: $\mathcal{X} \leftarrow MLP(\mathbf{P}')$ ▷ Learn the $K \times K$ $\mathcal{X}$-transformation matrix
5: $\mathbf{F}_\mathcal{X} \leftarrow \mathcal{X} \times \mathbf{F}_*$ ▷ Weight and permute $\mathbf{F}_*$ with the learnt $\mathcal{X}$
6: $\mathbf{F}_p \leftarrow \text{Conv}(\mathbf{K}, \mathbf{F}_\mathcal{X})$ ▷ Finally, typical convolution between $\mathbf{K}$ and $\mathbf{F}_\mathcal{X}$

- PointNet, PointNet++, DGCNN are all based on symmetric functions.
- In PointCNN, the authors attempt to permute the k-nearest neighborhood points into a canonical order with learned permutation matrix $\mathcal{X}$ and apply regular 1D convolution kernels to aggregate the neighbor features.

PointCNN [Li *et al.*, NeurIPS 2018]

PointConv

Dynamic weights for neighborhood points

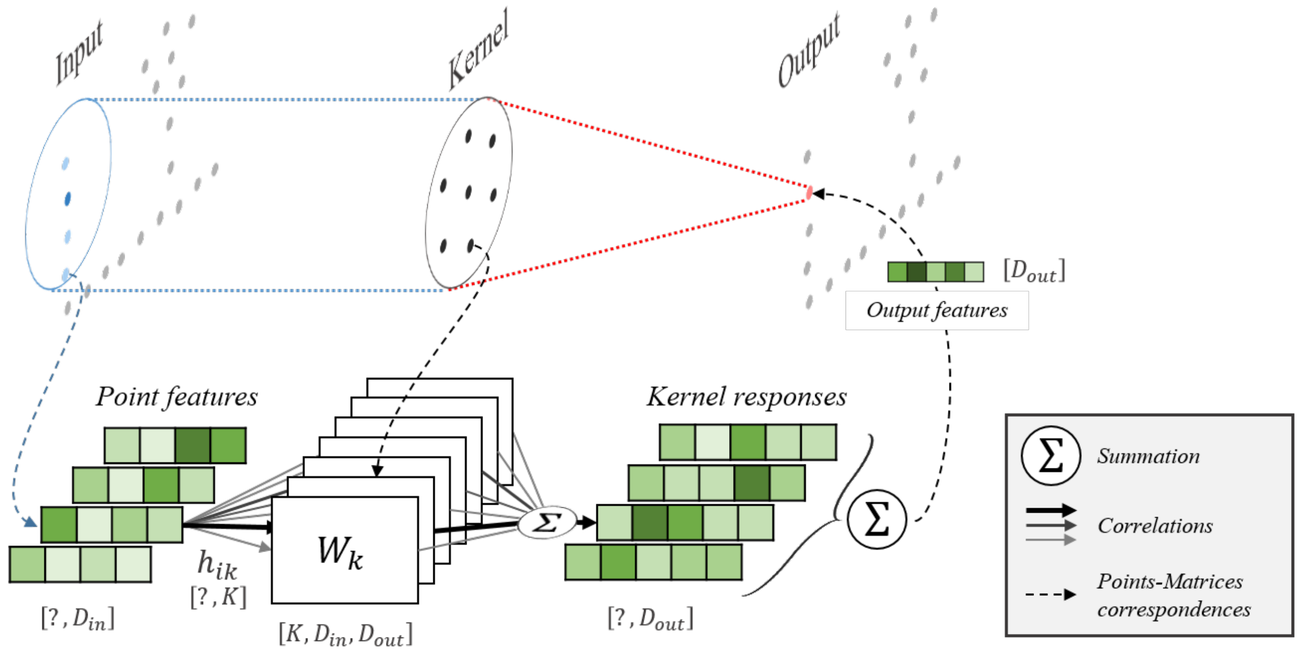


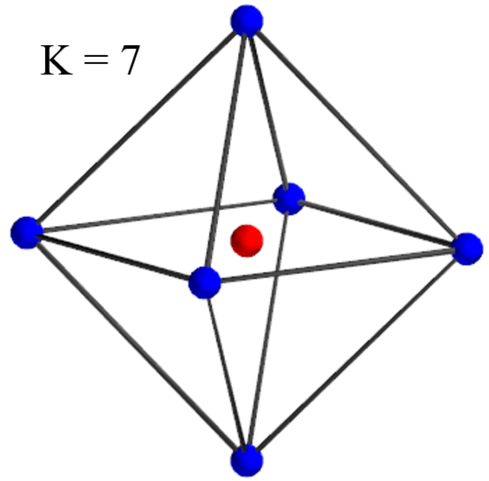
- PointCNN uses **static weight** and learns **dynamic permutation** for the neighborhood points. PointConv learns **dynamic weights** according to feature differences and point densities, while still uses symmetric functions to perform the reduction.

PointConv: Deep Convolutional Networks on 3D Point Clouds [Wu *et al.*, CVPR 2019]

KPConv

Interpolated dynamic weights for neighborhood points

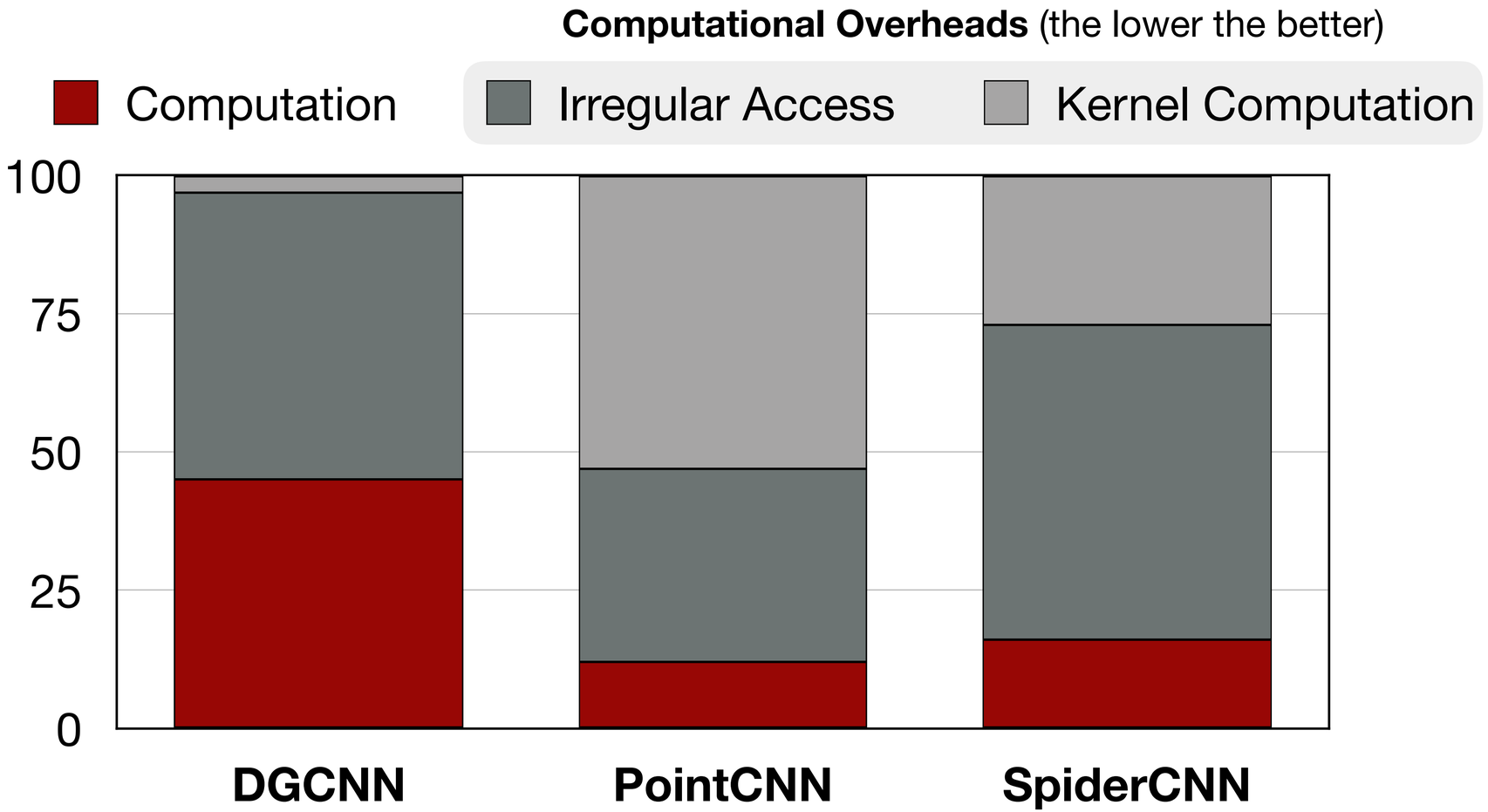




- KPConv is similar to PointConv. The main difference is that the **dynamic weights** are generated through **interpolation** from **kernel weights**, instead of using MLP layers. Thus, the weight generation process is far more efficient compared with KPConv.
- Right: 6 equally-distance **kernel points** on a sphere with given radius (the neighborhood radius) and the sphere center. The kernel for each neighborhood point is a distance-weighted sum of **kernel weights**.

KPConv: Flexible and Deformable Convolution for Point Clouds [Thomas *et al*., ICCV 2019]

Bottlenecks

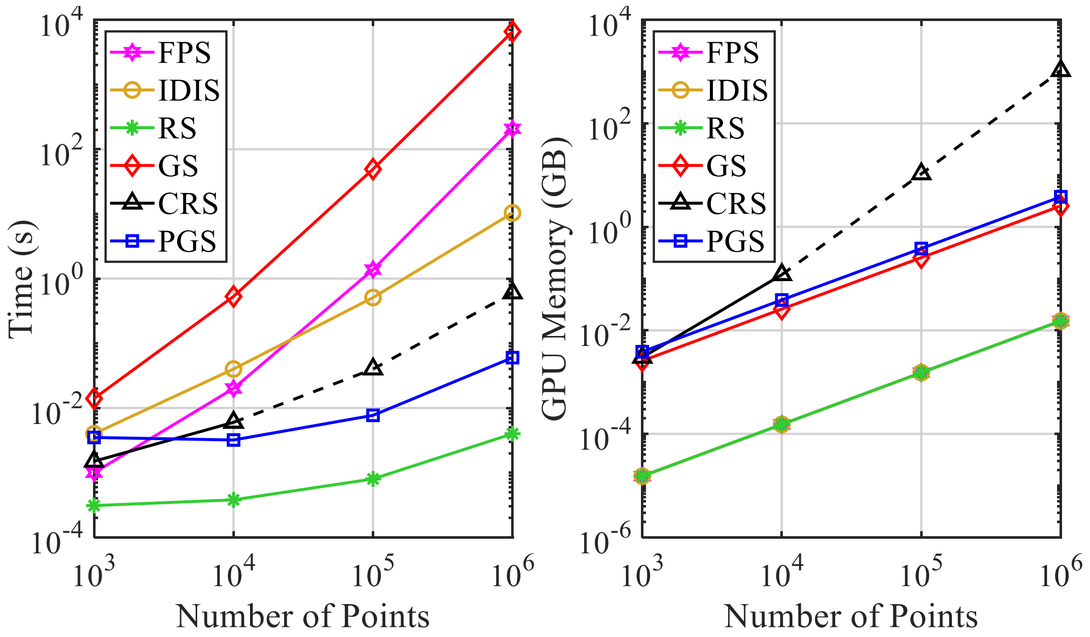


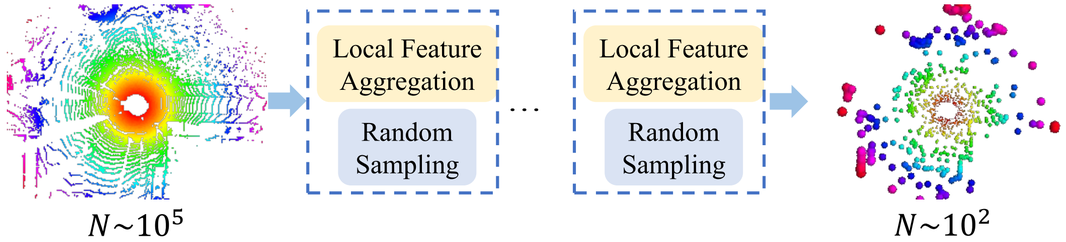
Up to **80% of the time** is wasted on **structuring the irregular data**, not on the actual feature extraction.

Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

Rand-LA Net

Improving the efficiency of point cloud convolution via fast downsampling





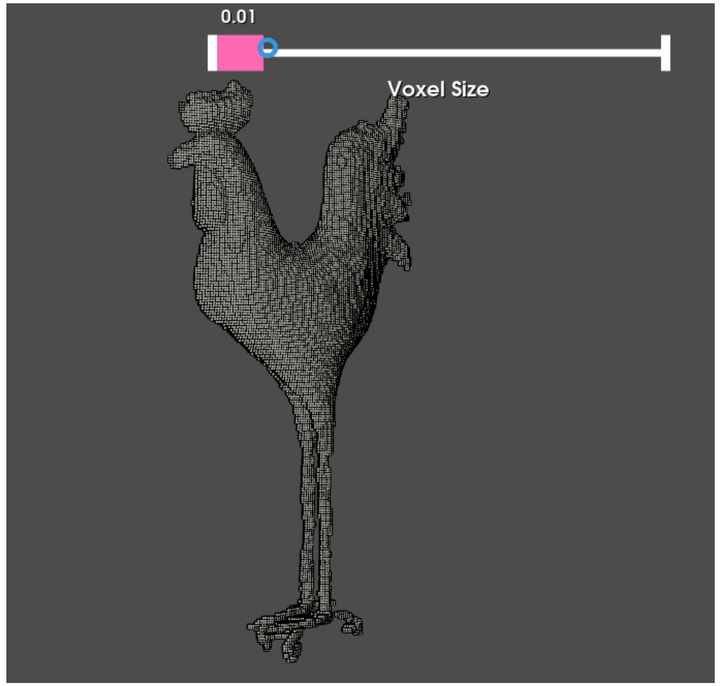
- Downsampling time can scale very quickly as number of points grows.
- Furthest-point sampling layers can be quite expensive in point cloud networks, random sampling can be a good approximation.

RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds [Hu *et al.*, CVPR 2020]

1.4 Sparse Voxels

Sparse Voxels

Store voxel grids as point clouds, not dense 4D tensor

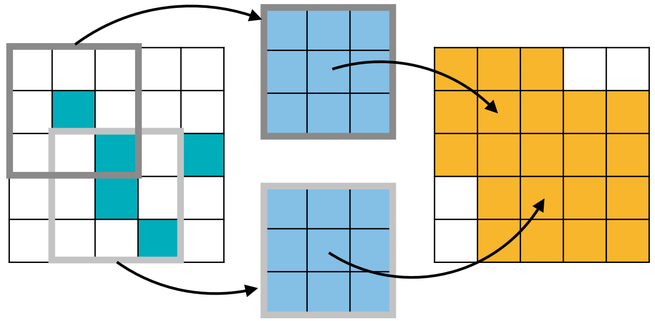


- **Dense voxels:**
- We create a 4D tensor V of size $H \times W \times D \times C$, where $[H, W, D]$ is the upper bound of voxelized coordinates. Memory cost is $H \times W \times D \times C$.
- **Sparse voxels:**
- Only quantize the coordinates of the point clouds, and still store the data sparsely. The memory cost is $N \times C$, where N is the number of points.
- **Why sparse voxels instead of point clouds?**
- Integer coordinates can make neighborhood query easier.

How to Voxelize Meshes and Point Clouds in Python [Nikolov, 2022]

Sparse convolution on sparse voxels

Conventional Convolution

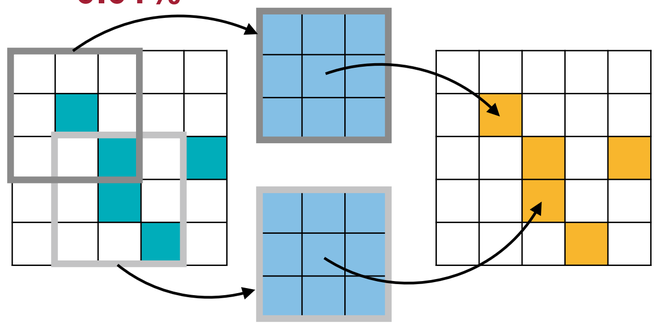


Input sparsity from ReLU

Nonzeros will dilate

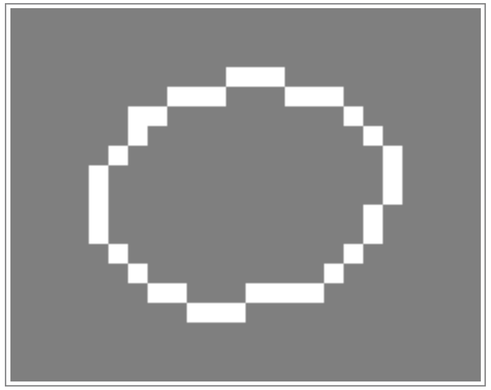
Sparse Convolution

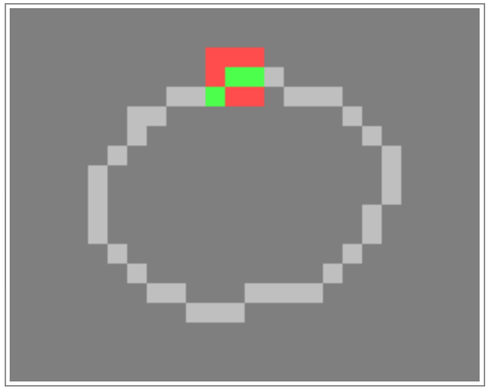
~0.01%



Input sparsity from the distribution in physical space

Nonzeros **will not** dilate





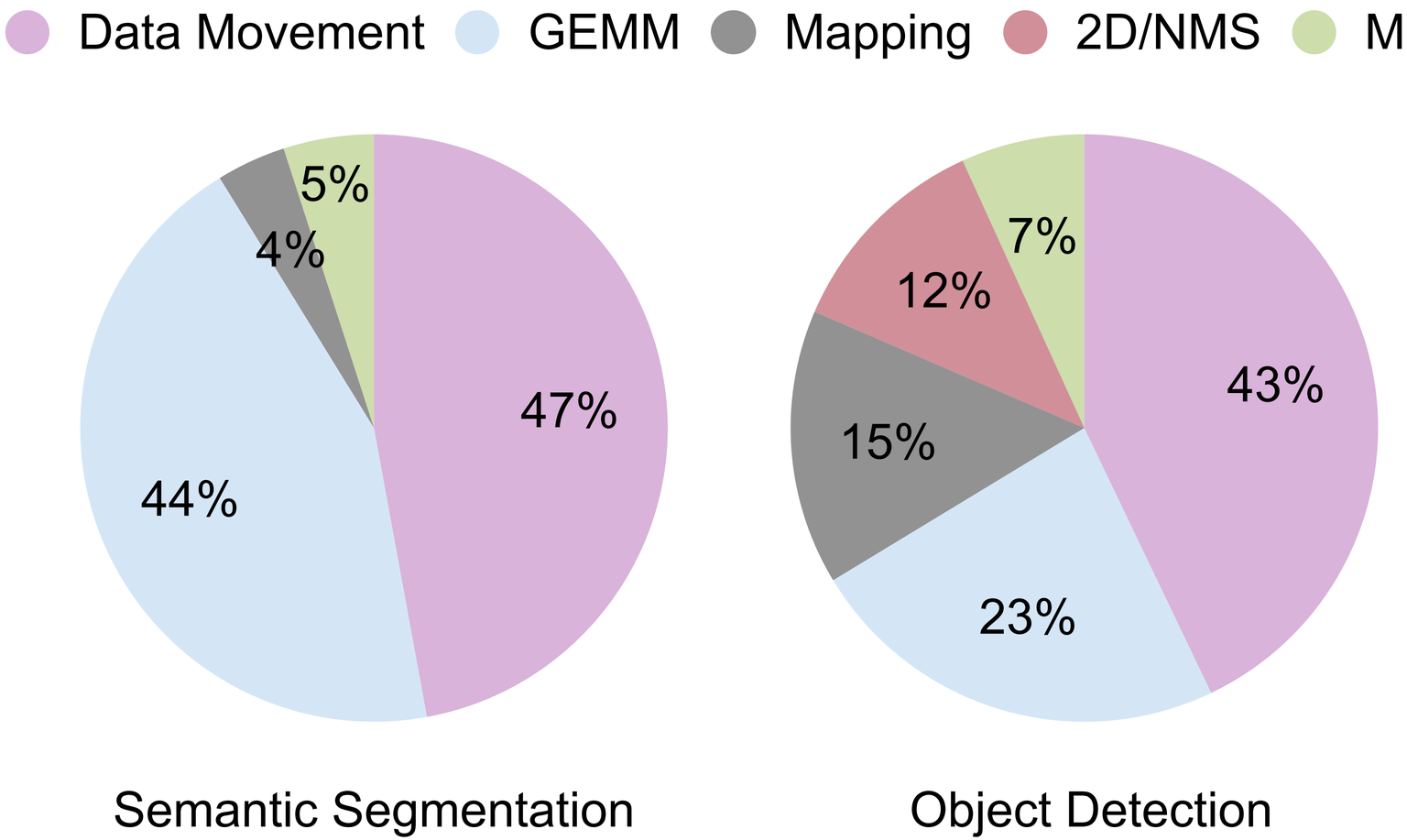
Submanifold Sparse Convolutional Neural Networks [Graham, BMVC 2015]

MinkowskiNet

Generalizing 2D CNNs to 3D sparse CNNs

Sparse convolution is inefficient on GPUs

Bottleneck is data movement and mapping due to sparsity and irregularity

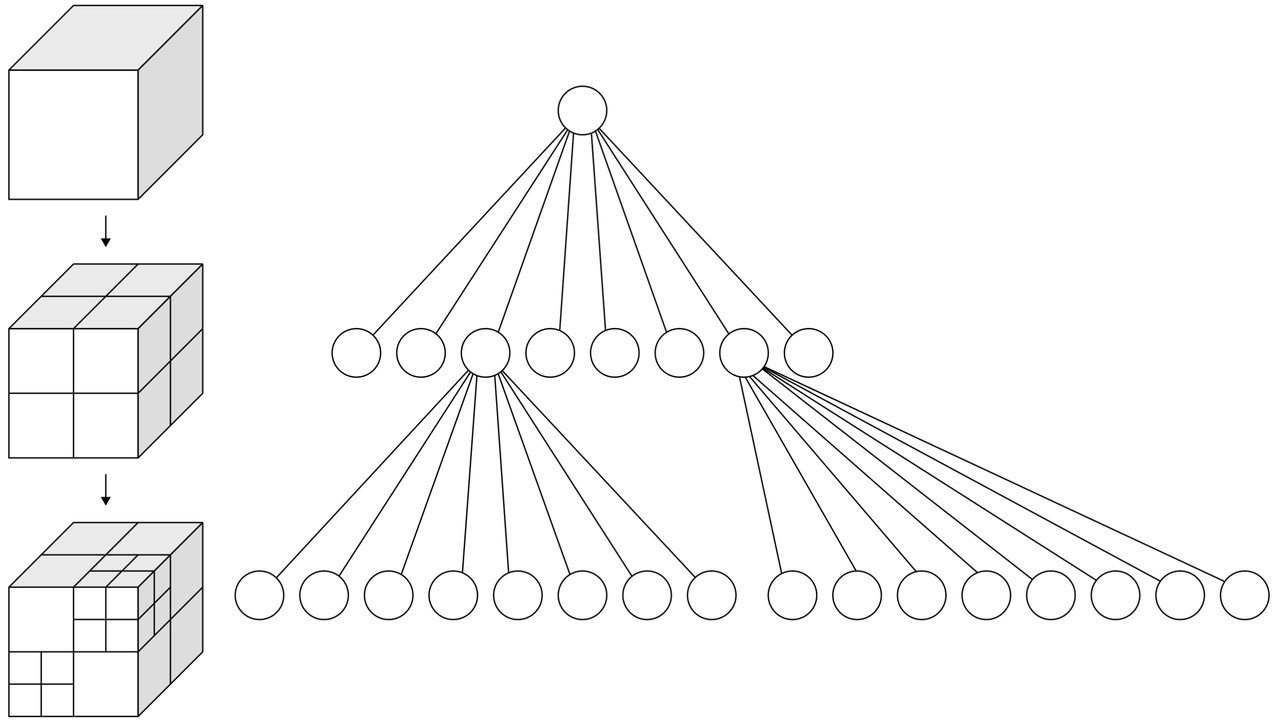


TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

1.5 Octree

Octree

A compact and recursive representation for 3D data

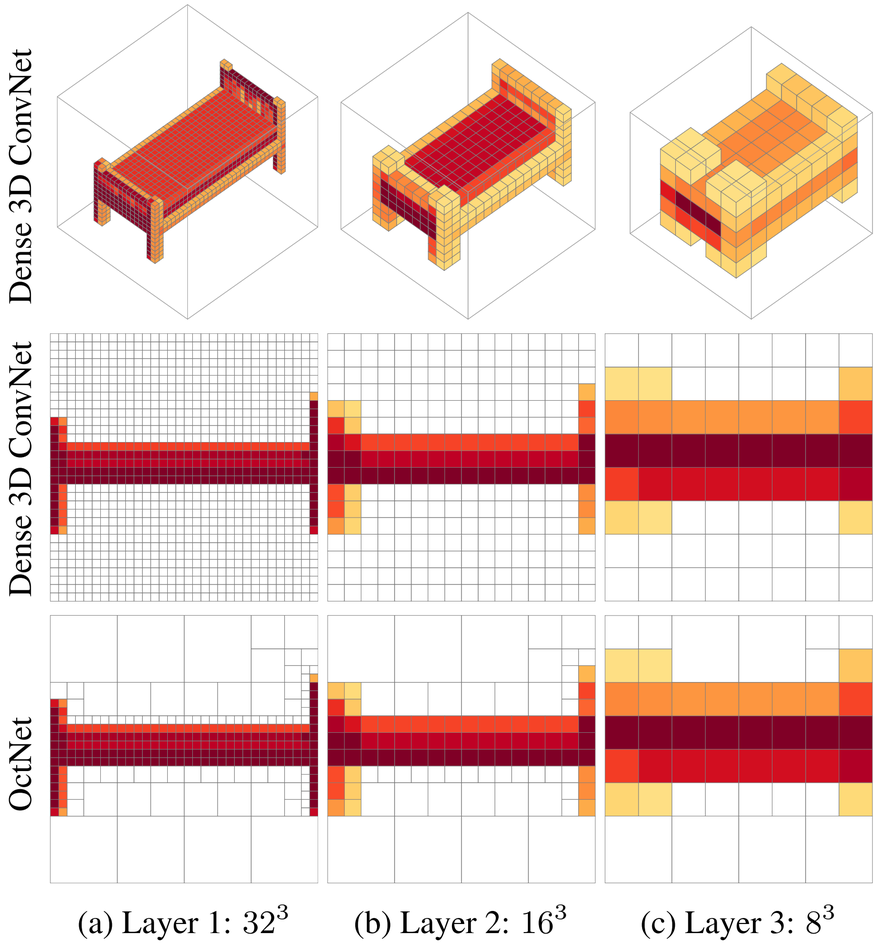


- Octree is a data structure that recursively divides the 3D space into finer granularity voxel grids. It has different voxel resolution at different spatial locations. Lower voxel resolution is used for less informative areas.

https://en.wikipedia.org/wiki/Octree

OctNet

Deep learning on Octrees



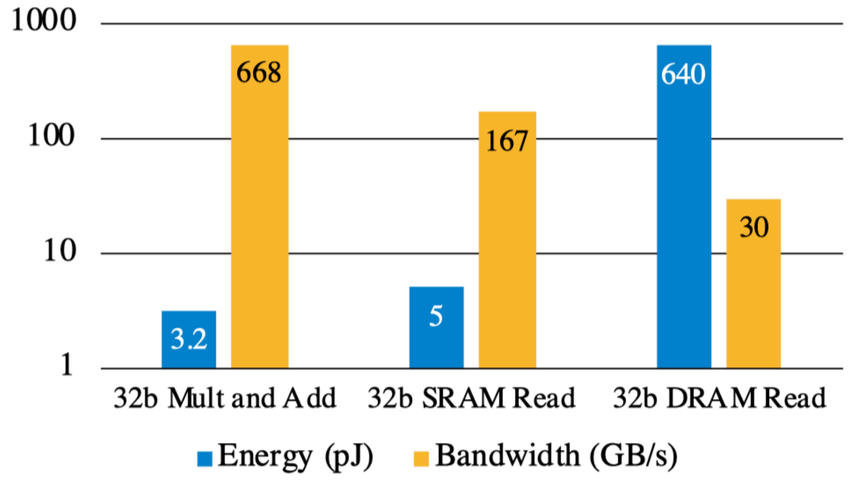
(a) Layer 1: 32^3 (b) Layer 2: 16^3 (c) Layer 3: 8^3

OctNet: Learning Deep 3D Representations at High Resolutions [Riegler *et al.*, CVPR 2017]
O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis [Wang *et al.*, ToG 2018]

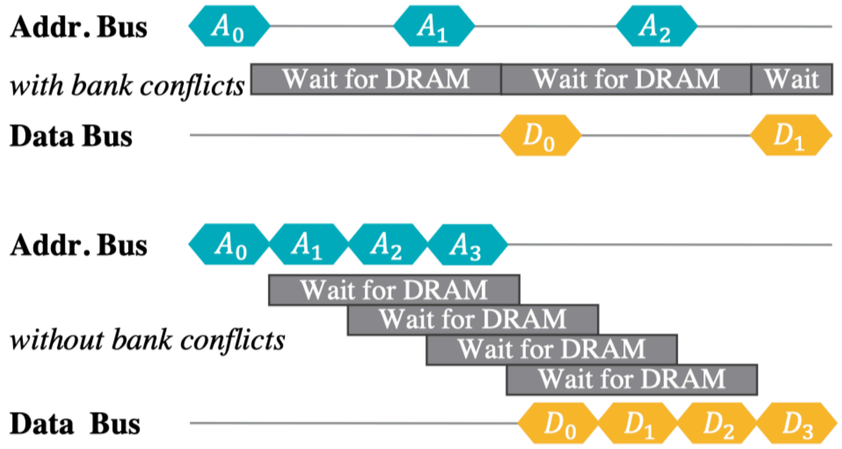
1.6 Hybrid

Point-Voxel CNN

Memory operations are expensive



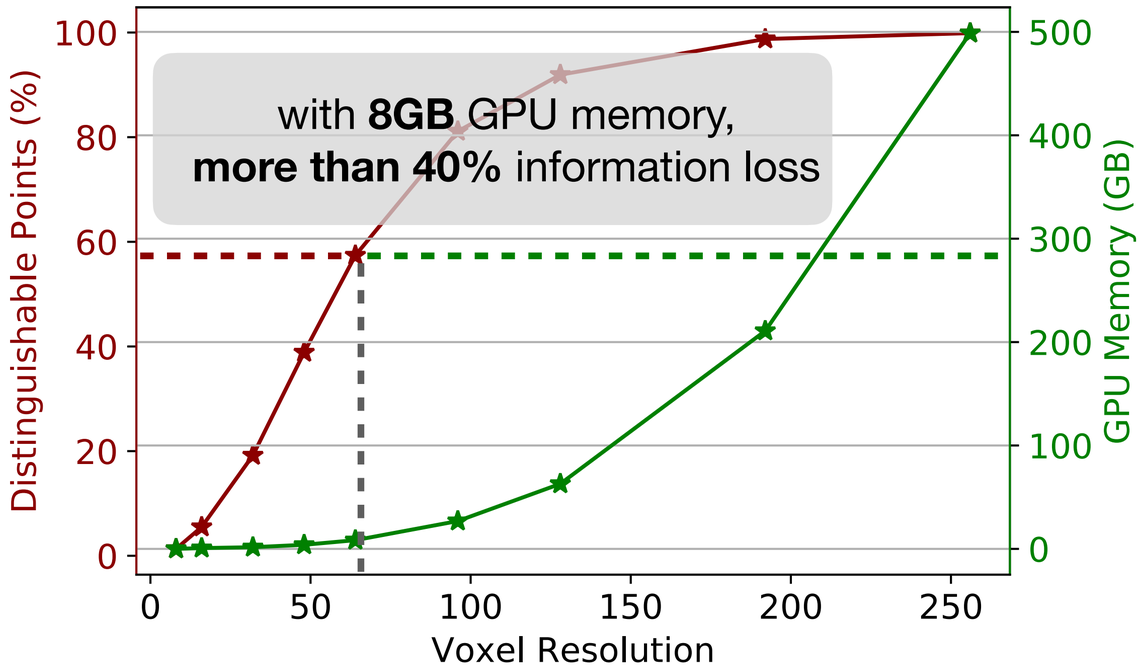
Off-chip DRAM access is much more expensive than arithmetic operation!

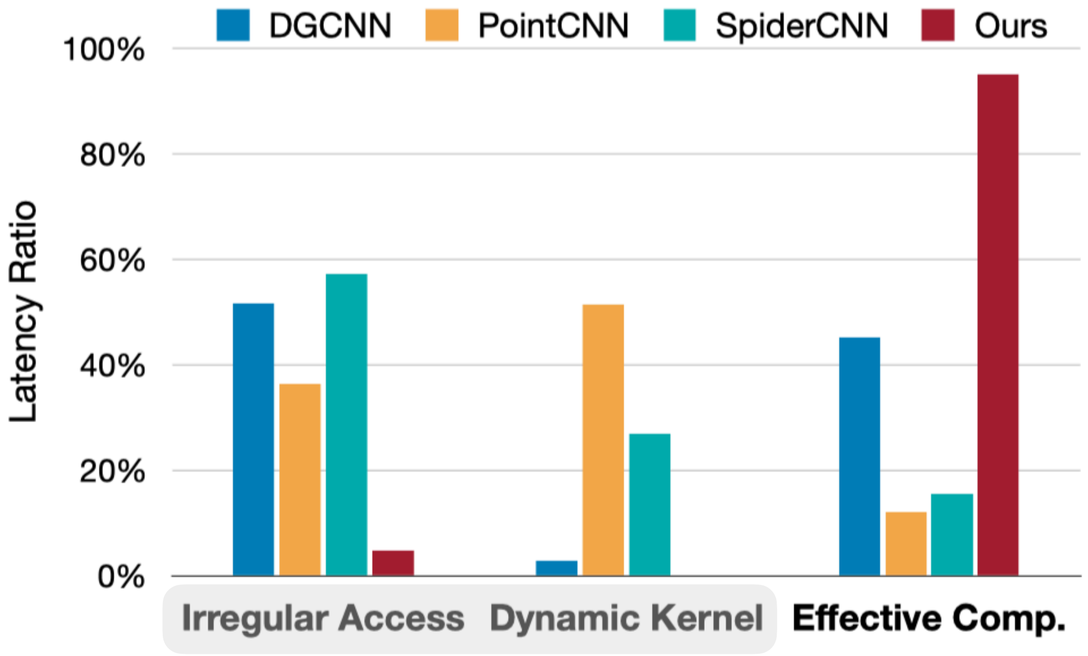


Random memory access is inefficient due to the potential bank conflicts!

Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

Bottlenecks of point and voxel-based methods

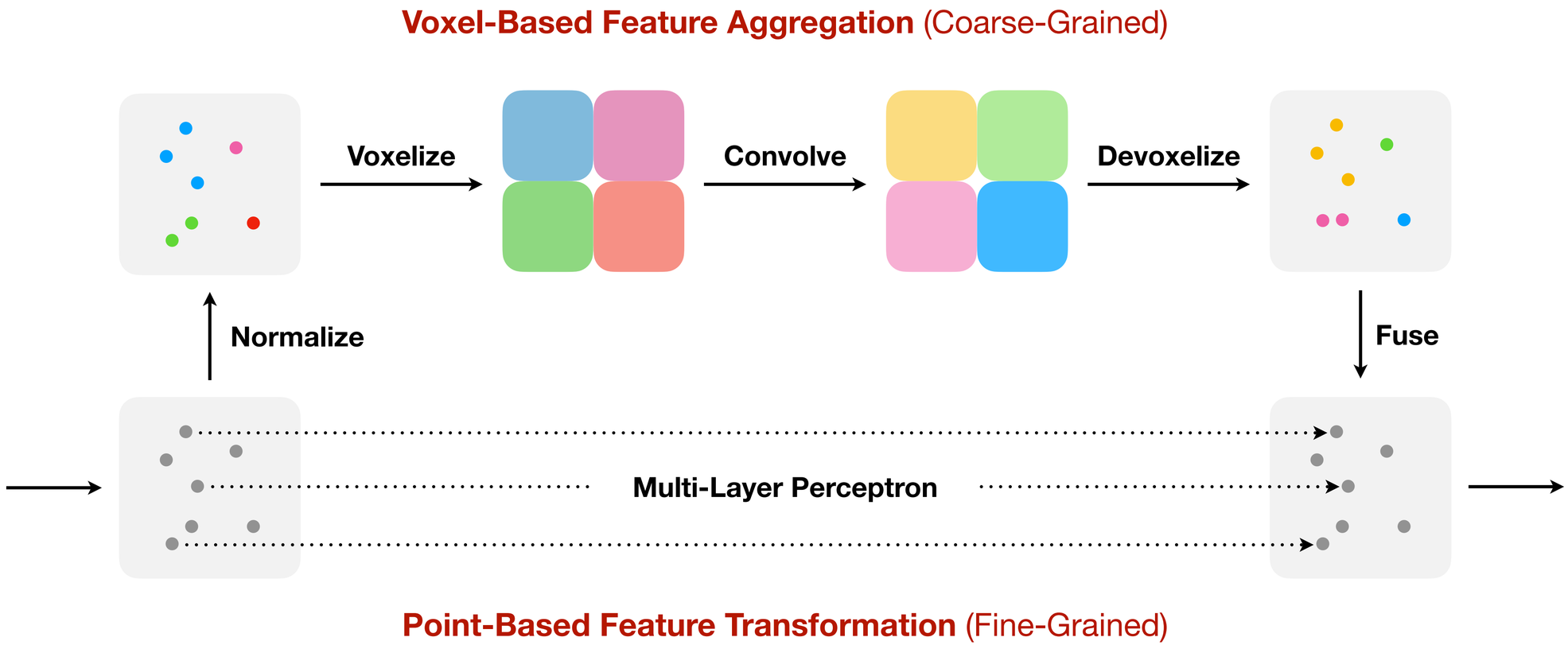




Computational Overheads
(the lower the better)

Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

Point-Voxel CNN

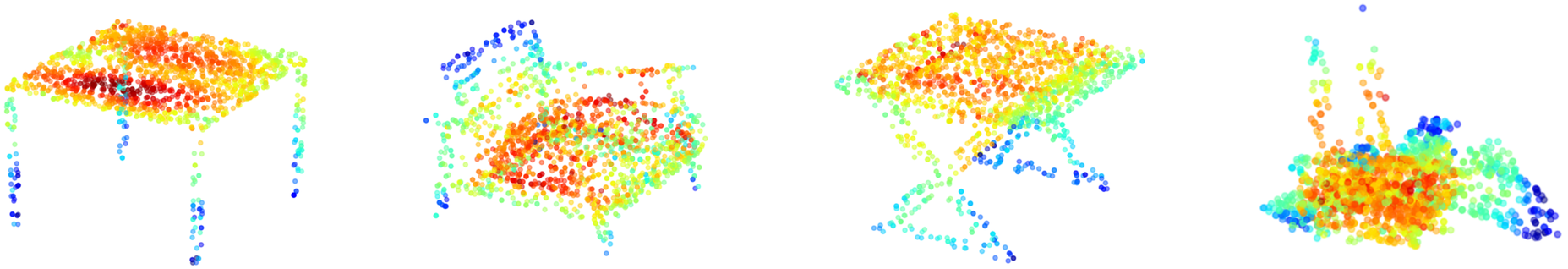


- Insights for PVCNN:
- Voxel-based branch has better regularity and no irregular access / dynamic kernel cost for convolution, it is a better choice for neighborhood information aggregation.
- Point-based branch can keep high resolution and alleviate the information loss in voxelization

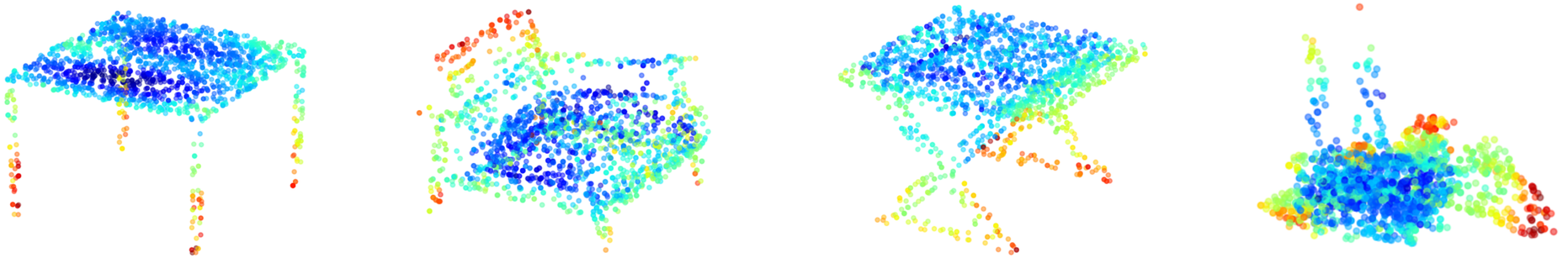
Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

Point-Voxel CNN

Features from **Voxel-Based Branch**:

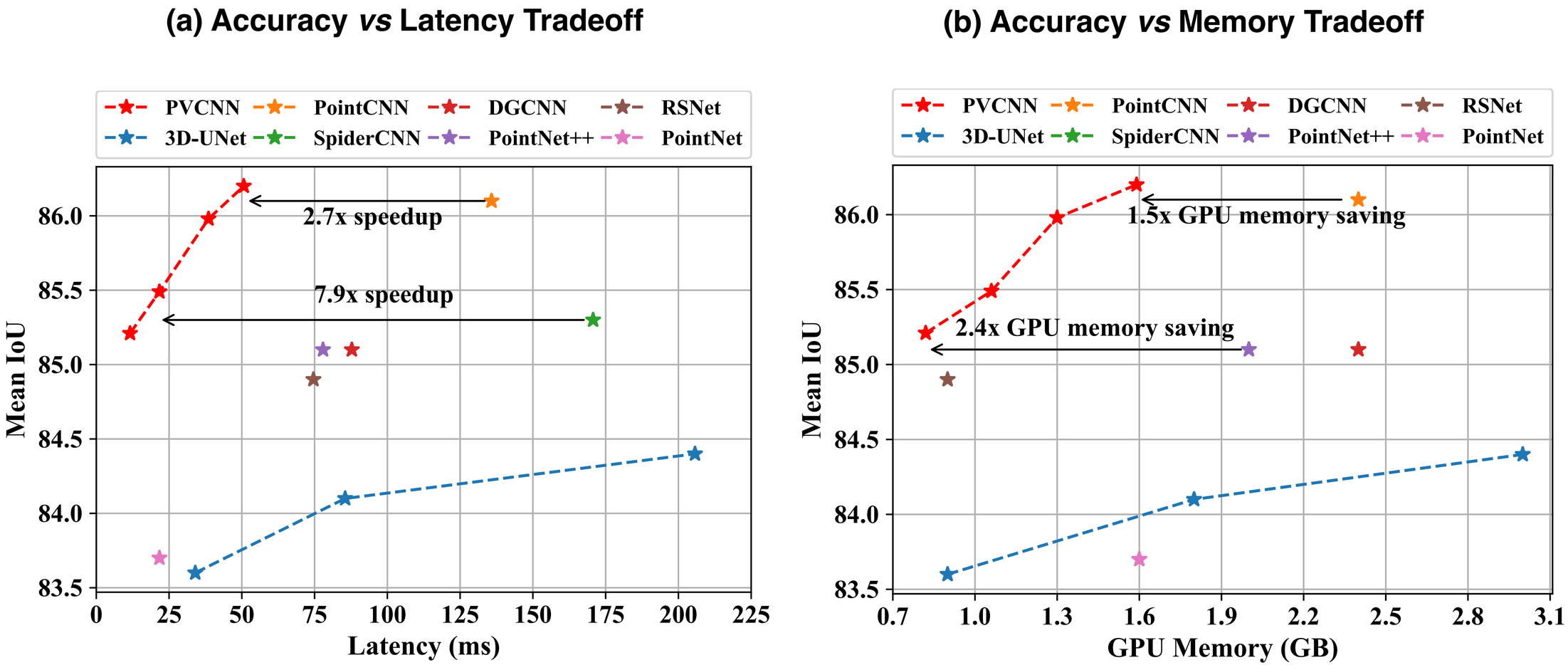


Features from **Point-Based Branch**:



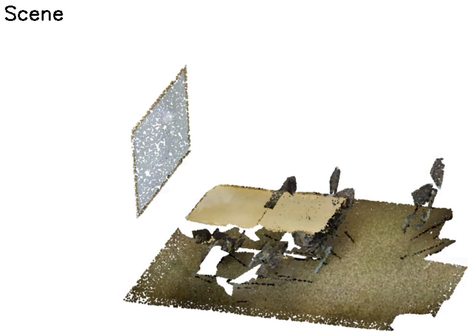
Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

Point-Voxel CNN is fast and accurate

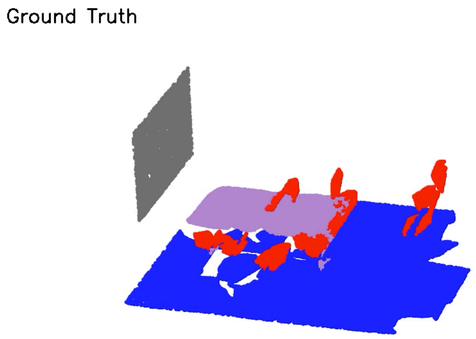


Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

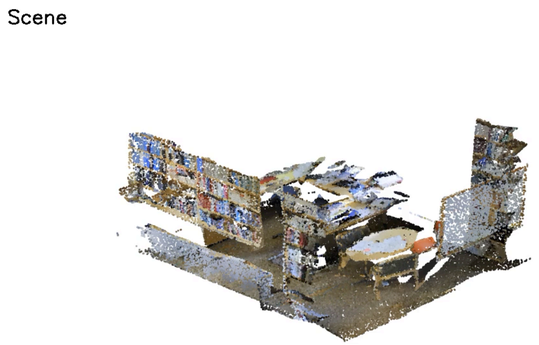
Demos for Indoor 3D Segmentation



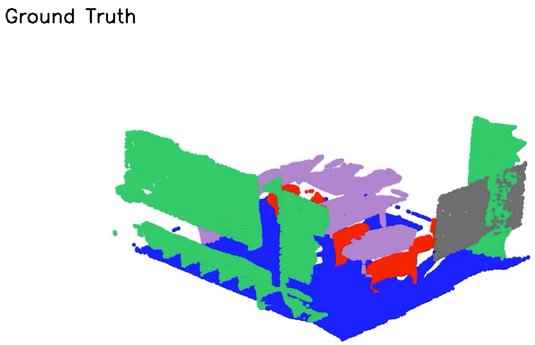
Input Scene



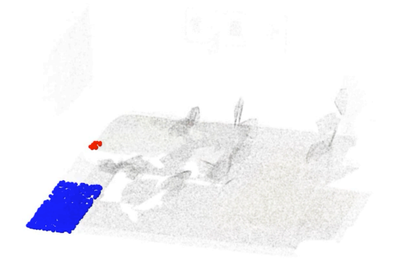
Ground Truth



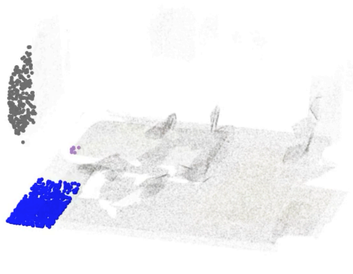
Input Scene



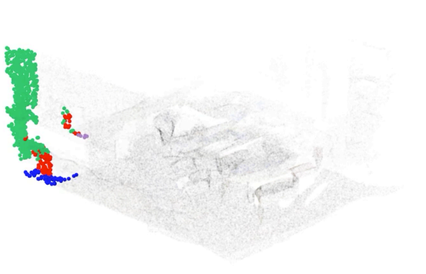
Ground Truth



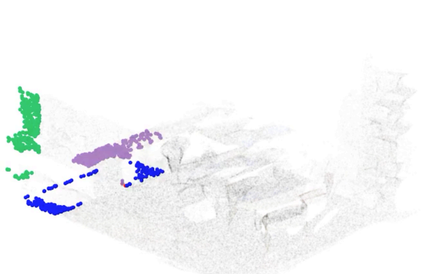
PointNet
GPU Memory: **1.9 GB**
Time: **1.9 sec**



PVCNN (ours)
GPU Gemory: **1.2 GB**
Time: **1.0 sec**



PointNet
GPU Memory: **1.9 GB**
Time: **1.9 sec**

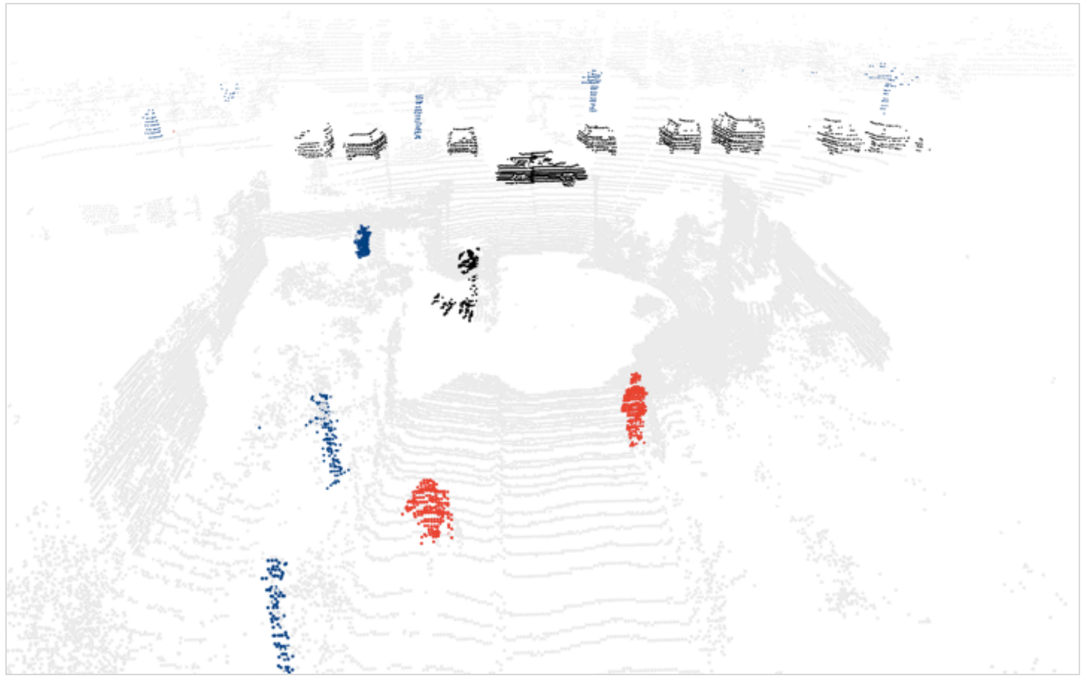


PVCNN (ours)
GPU Memory: **1.2 GB**
Time: **1.0 sec**

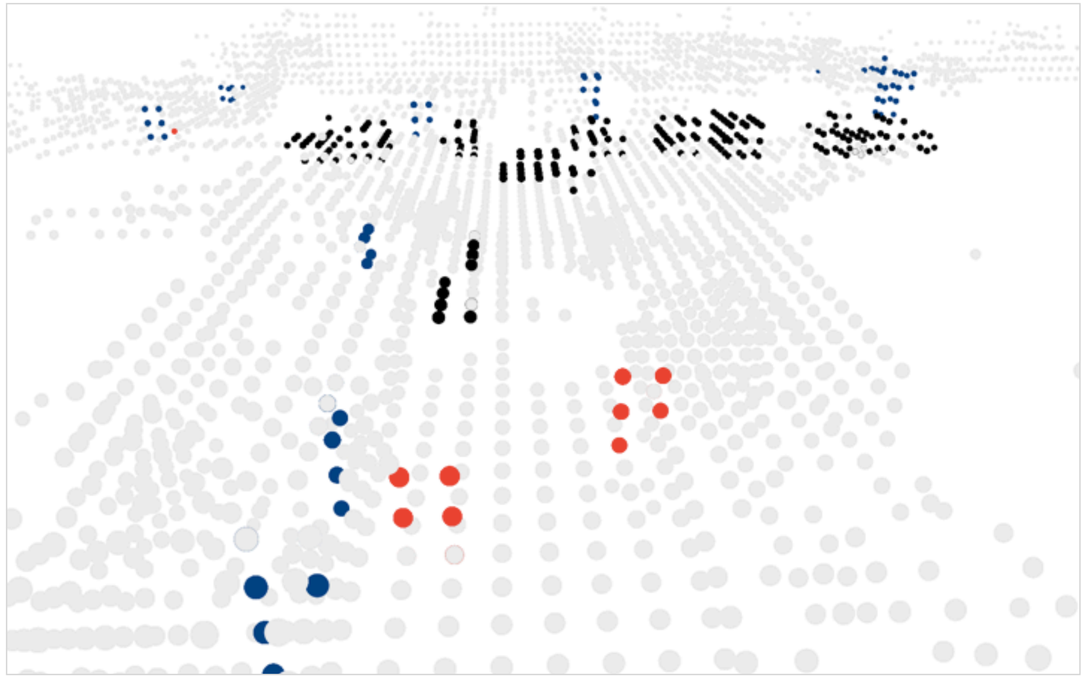
Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

Limitation of Point-Voxel Convolution

Voxelization in PVConv can introduce information loss



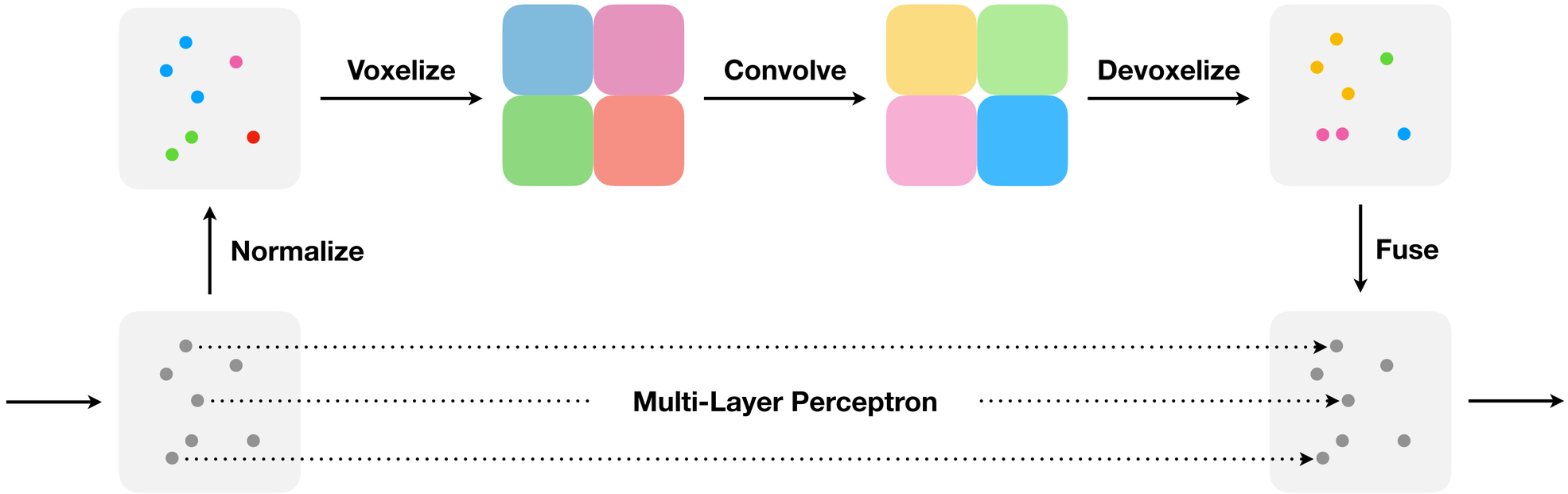
Input Scene



Voxelized Scene

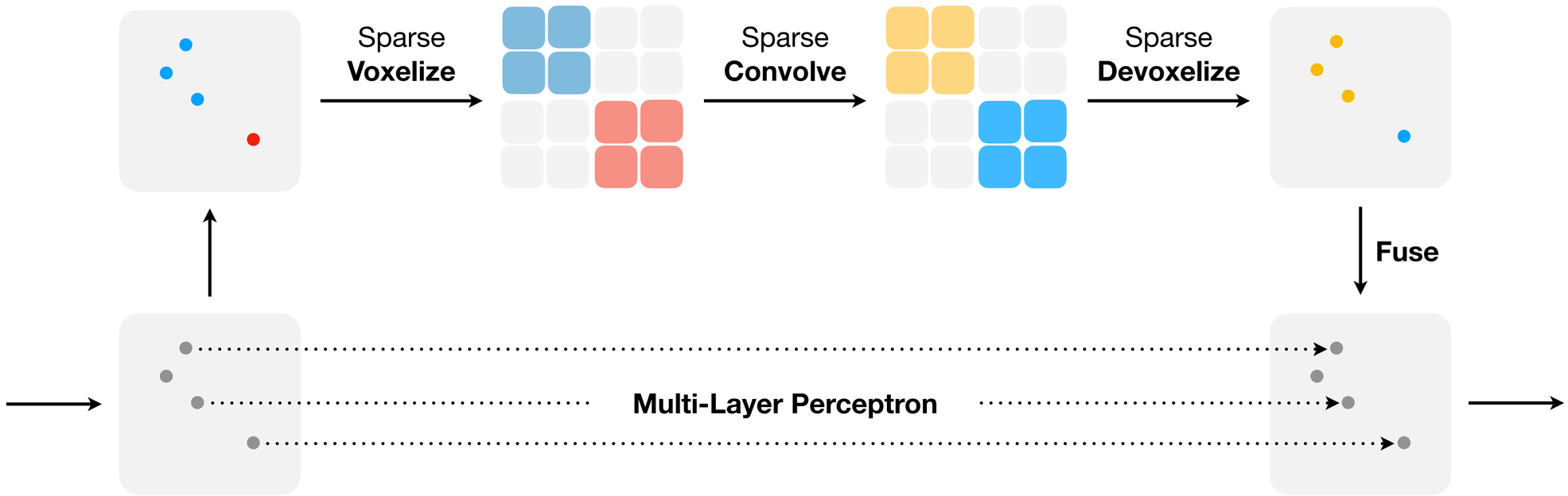
Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang *et al.*, ECCV 2020]

Point-Voxel Convolution (PVConv)



Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]

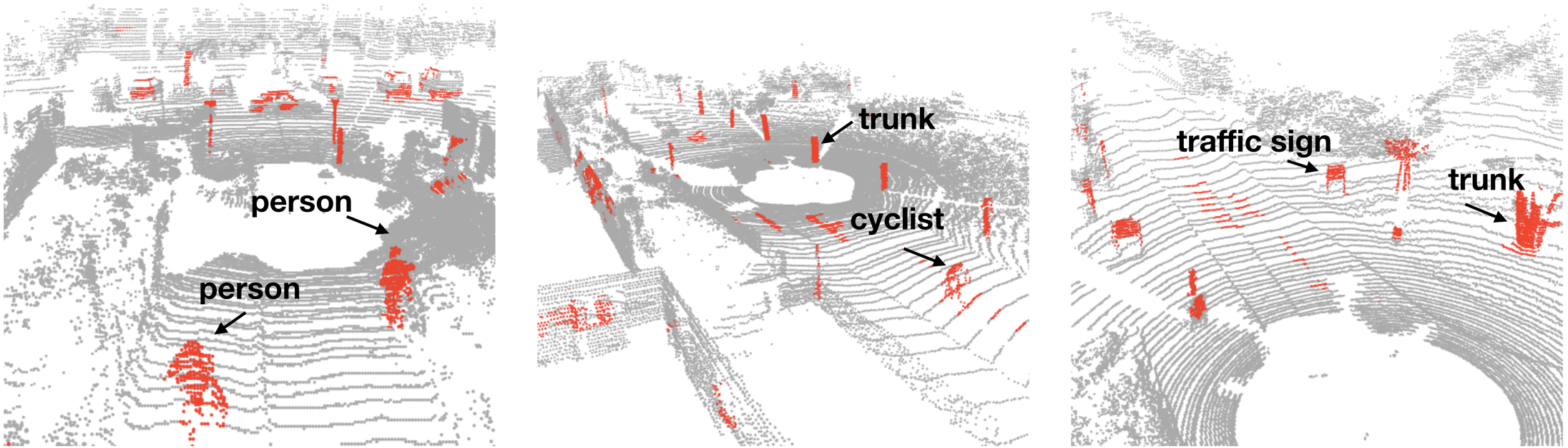
Sparse Point-Voxel Convolution (SPVConv)



Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang *et al.*, ECCV 2020]

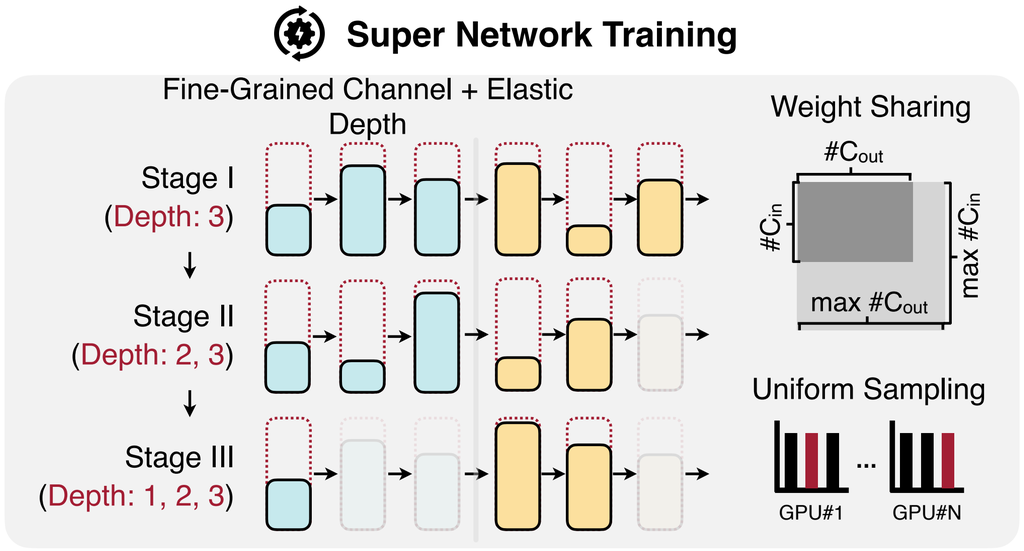
Sparse Point-Voxel Convolution (SPVConv)

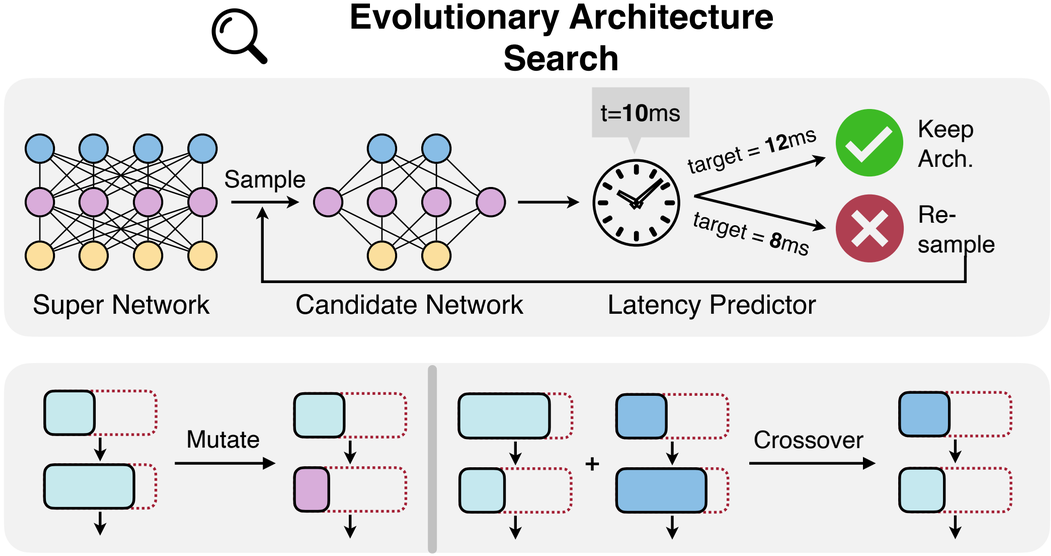
Features from **Point-Based Branch**:

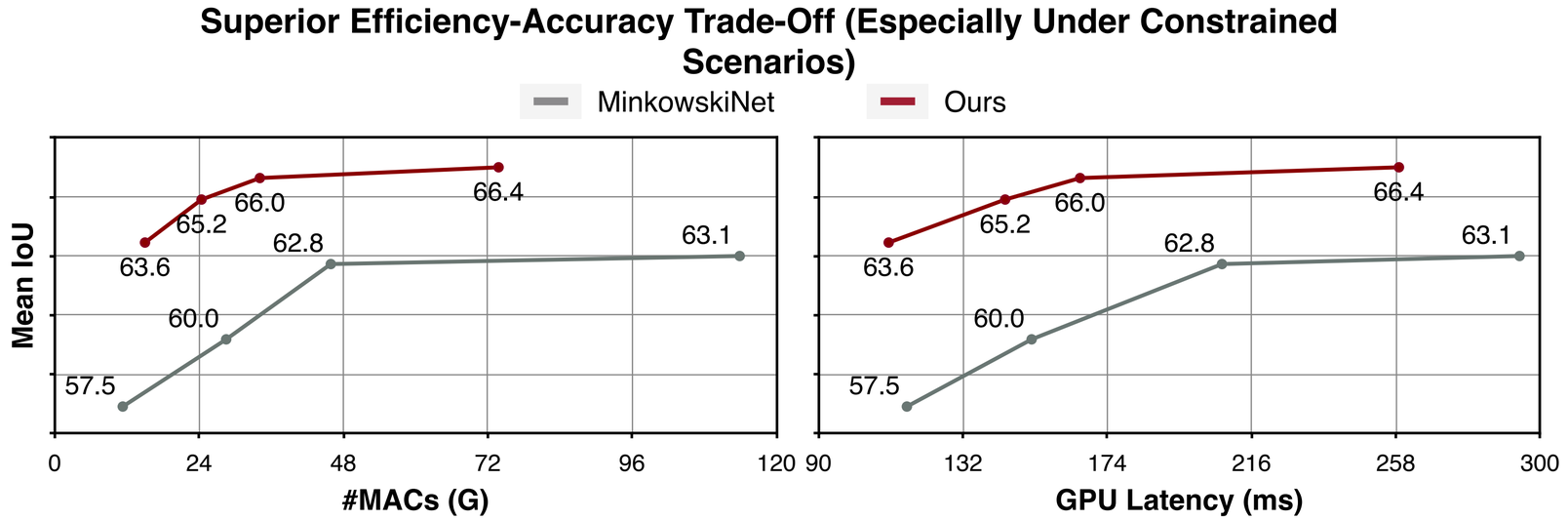


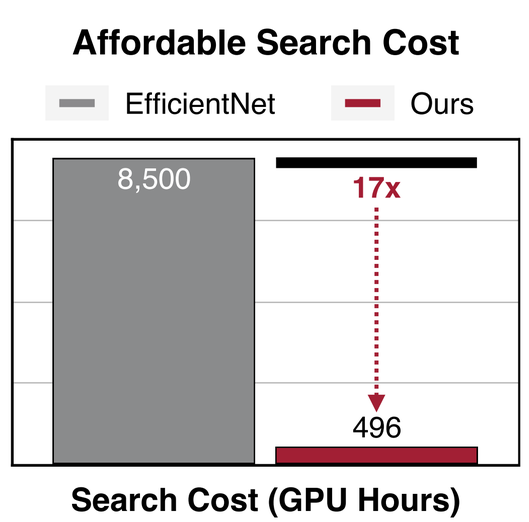
Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang *et al.*, ECCV 2020]

3D Neural Architecture Search with SPVConv





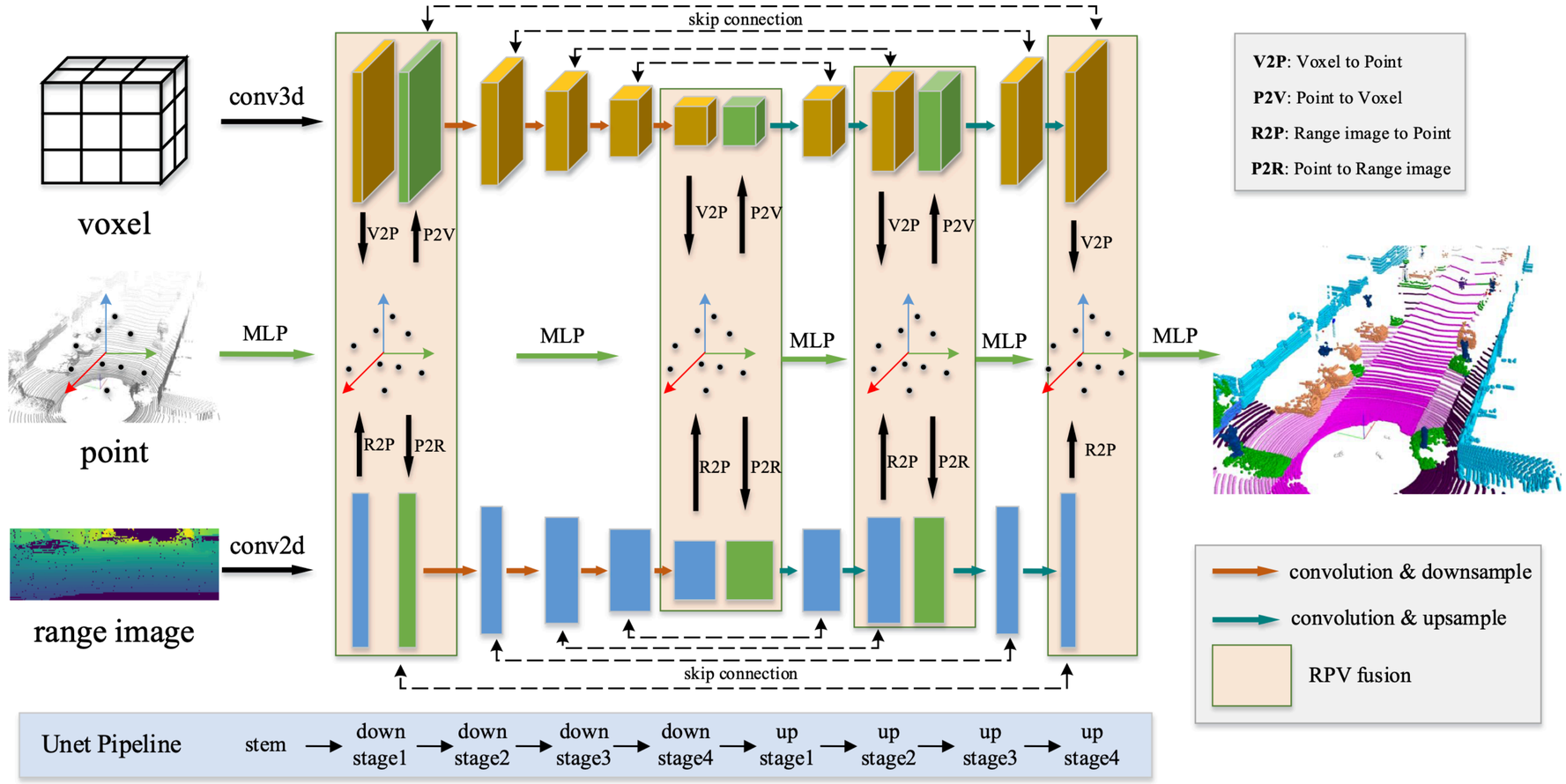




Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang *et al.*, ECCV 2020]

Range-Point-Voxel Convolution (RPVConv)

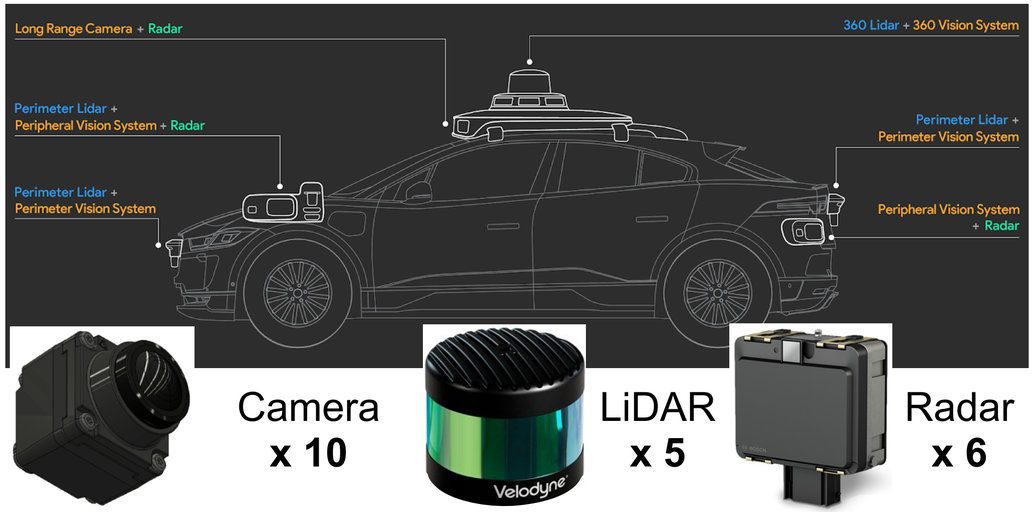
Adding range image inputs to SPVConv



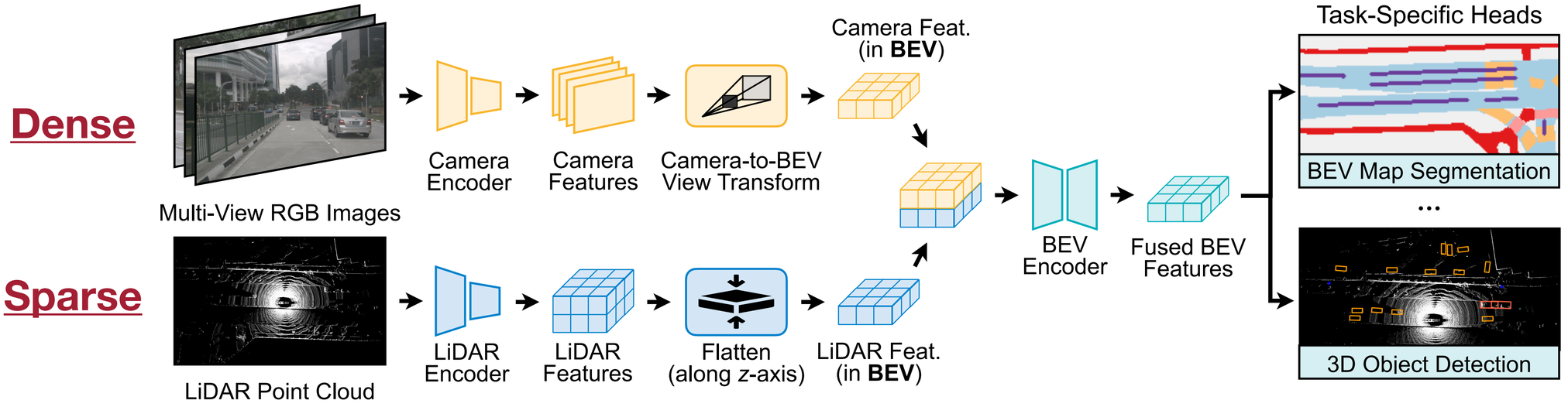
RPVNet: A Deep and Efficient Range-Point-Voxel Fusion Network for LiDAR Point Cloud Segmentation [Xu *et al.*, ICCV 2021]

Efficient Point Cloud Recognition and Perception

BEVFusion: multi-task, multi-sensor fusion



- **Multiple tasks**: detecting the vehicles and pedestrians, segmenting the lanes and drivable regions, *etc*
- **Multiple sensors**: camera produces ***dense*** images, and LiDAR produces ***sparse*** point clouds
- Two branches of dataflow: one ***dense*** branch for camera, one ***sparse*** branch for LiDAR



BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation [Liu *et al.*, arXiv 2022]

BEVFusion: Multi-Task Multi-Sensor Fusion

BEVFusion takes **multi-modal** sensory inputs and supports **multiple** 3D perception tasks.

Multi-View Camera

LiDAR

https://youtu.be/uCAka90si9E

BEVFusion ranks 1st on Waymo leaderboard

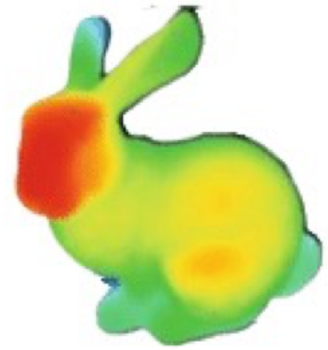
As of Nov 9th, 2022

	Method Name	Object Type	Sensors	Frames [-p, +f]	Latency (s)	AP / L1	APH / L1	AP / L2	APH / L2	Date (Pacific Daylight Time)
		ALL_NS ✖ ▾	All ▾		Show all ▾					
1	BEVFusion-TTA	ALL_NS	CL	[-2, +0]		0.8604	0.8476	0.8122	0.7997	2022-09-18 21:46
2	LidarMultiNet-TTA	ALL_NS	L	[-2, +0]		0.8605	0.8472	0.8124	0.7994	2022-09-28 10:08
3	MPPNetEns-MMLab	ALL_NS	L	[-15, +0]		0.8548	0.8414	0.8091	0.7960	2022-09-02 13:57
4	3DAM_Ens-Shanghai AI Lab	ALL_NS	L	[-4, +0]		0.8528	0.8378	0.8065	0.7919	2022-07-19 01:40
5	LIVOX_Detection	ALL_NS	L	[-6, +0]		0.8482	0.8354	0.8022	0.7896	2022-05-10 21:18
6	MT3D	ALL_NS	L	[-3, +0]		0.8503	0.8367	0.8006	0.7873	2022-06-15 05:31
7	MT-Net	ALL_NS	L	[-2, +0]		0.8470	0.8322	0.7989	0.7845	2022-07-10 22:27
8	DeepFusion-Ens	ALL_NS	CL	[-4, +0]		0.8437	0.8322	0.7954	0.7841	2022-03-15 07:59
9	3dal-ens	ALL_NS	L	[--4, +0]		0.8463	0.8309	0.7968	0.7820	2022-07-02 02:08
10	InceptioLidar	ALL_NS	L	[-9, +0]		0.8380	0.8246	0.7915	0.7784	2022-02-28 23:09

BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation [Liu *et al.*, arXiv 2022]

Revisiting 3D Representations

Sparse convolution is dominant in real-world applications.

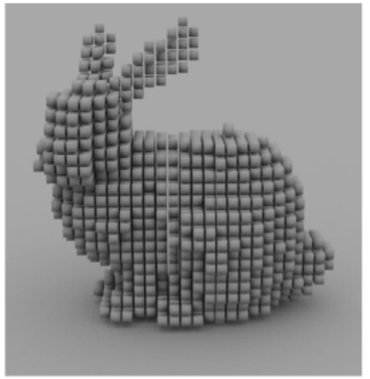


1.1 Multi-view images: Losing information

Projection

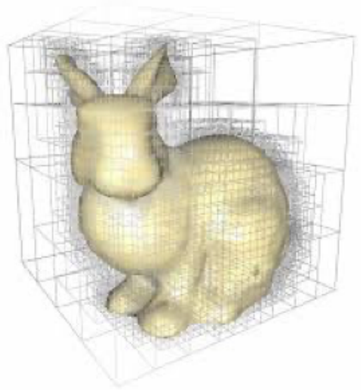
1.3 Point Cloud: Irregular access and dynamic kernel overhead

Voxelize



1.2 Voxels: Not scalable
1.4 Sparse voxels: high accuracy, good scalability, widely used, but **good system support needed**!

Voxelize

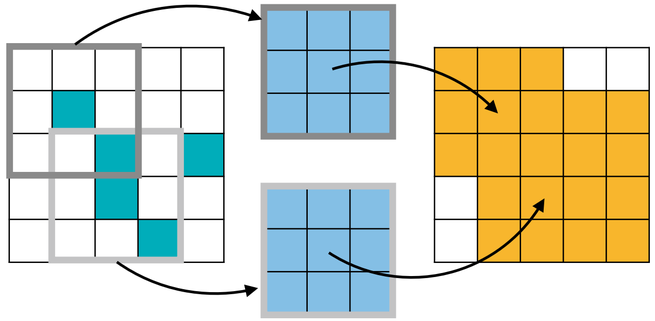


1.5 Octree: challenging to define convolution on octrees.
Not widely used recently.

2. System Support

Sparse convolution on sparse voxels

Conventional Convolution

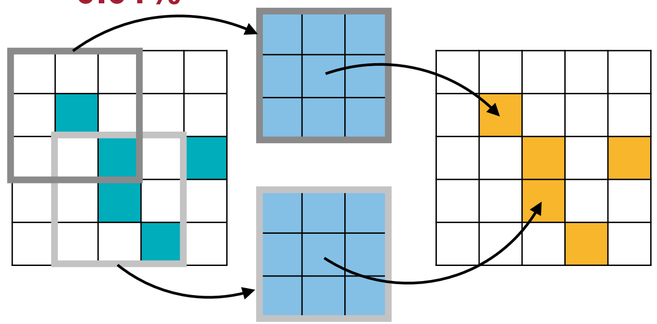


Input sparsity from ReLU

Nonzeros will dilate

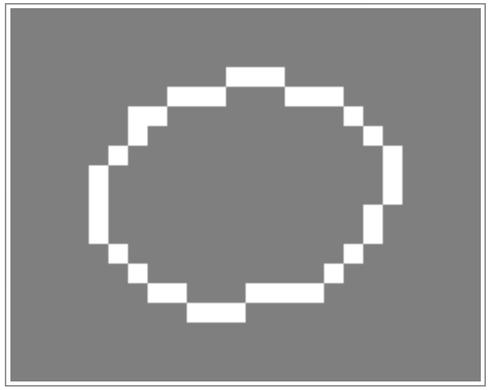
Sparse Convolution

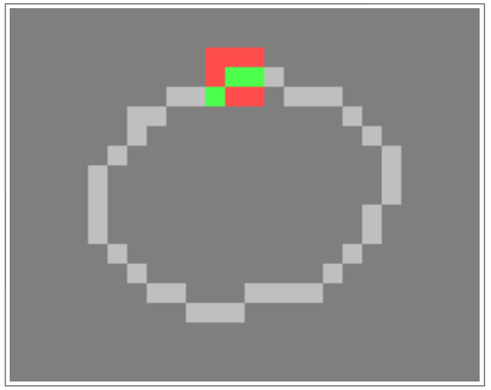
~0.01%



Input sparsity from the distribution in physical space

Nonzeros **will not** dilate



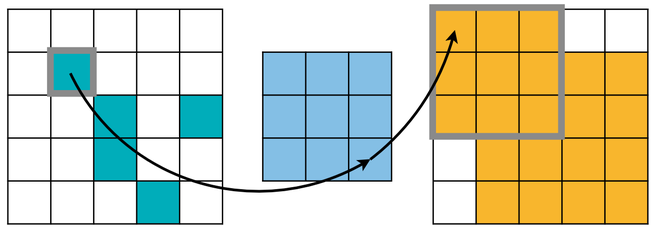


Submanifold Sparse Convolutional Neural Networks [Graham, BMVC 2015]

Sparse convolution computation

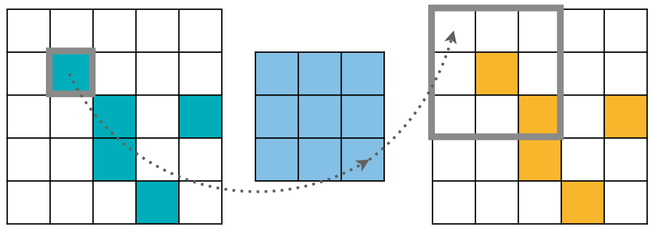
A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



(P_0, Q_0, $W_{1,1}$)

Sparse Convolution



No compute

Maps
(In, Out, Wgt)

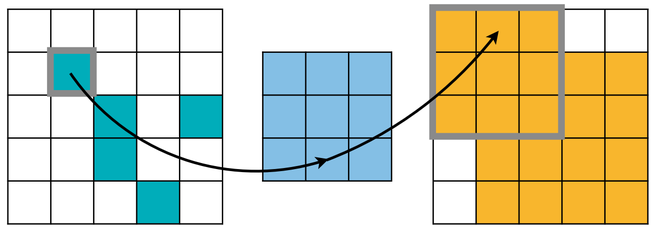
Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Sparse convolution computation

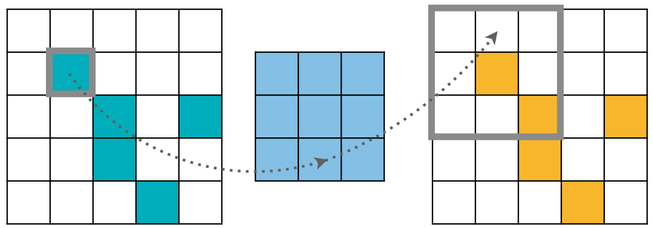
A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



(P_0, Q_0, $W_{1,1}$)
(P_0, Q_1, $W_{1,0}$)

Sparse Convolution



No compute
No compute

Maps
(In, Out, Wgt)

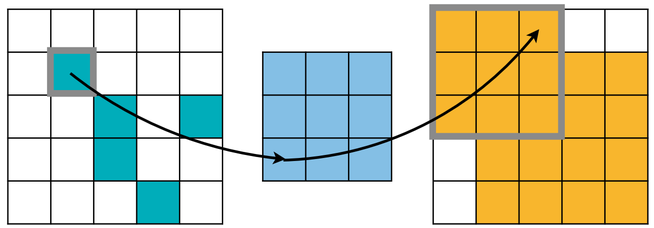
Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

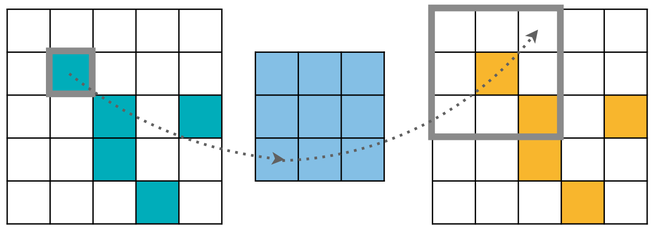
Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In, Out, Wgt)

(P_0, Q_0, $W_{1,1}$)
(P_0, Q_1, $W_{1,0}$)
(P_0, Q_2, $W_{1,-1}$)

No compute
No compute
No compute

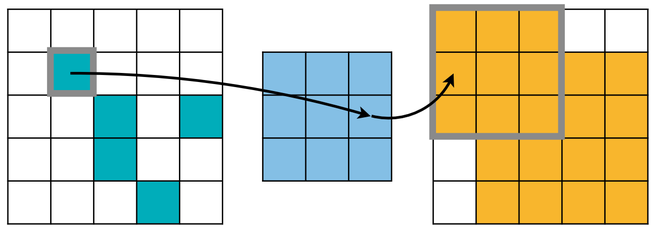
Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

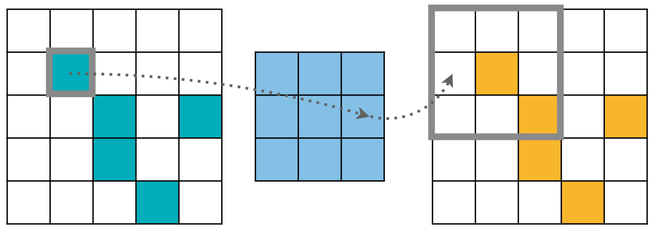
Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In, Out, Wgt)

$(P_0, Q_0, W_{1,1})$
$(P_0, Q_1, W_{1,0})$
$(P_0, Q_2, W_{1,-1})$
$(P_0, Q_3, W_{0,1})$

No compute
No compute
No compute
No compute

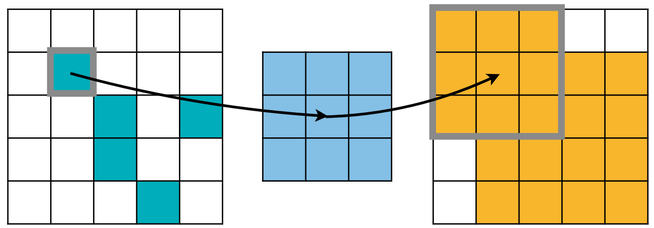
Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

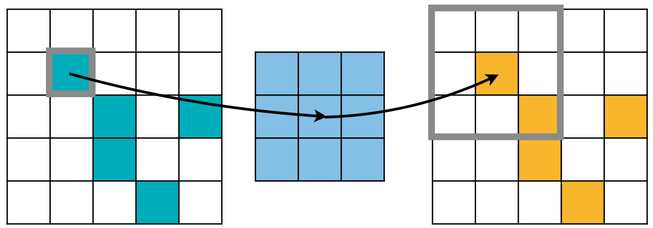
Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In, Out, Wgt)

$(P_0, Q_0, W_{1,1})$
$(P_0, Q_1, W_{1,0})$
$(P_0, Q_2, W_{1,-1})$
$(P_0, Q_3, W_{0,1})$
$(P_0, Q_4, W_{0,0})$

No compute
No compute
No compute
No compute
$(P_0, Q_0, W_{0,0})$

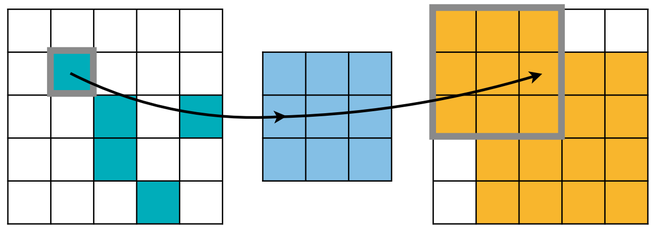
Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

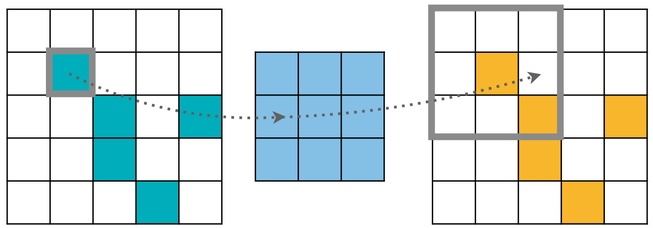
Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In, Out, Wgt)

Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

(P_0, Q_0, $W_{1,1}$)
(P_0, Q_1, $W_{1,0}$)
(P_0, Q_2, $W_{1,-1}$)
(P_0, Q_3, $W_{0,1}$)
(P_0, Q_4, $W_{0,0}$)
(P_0, Q_5, $W_{0,-1}$)

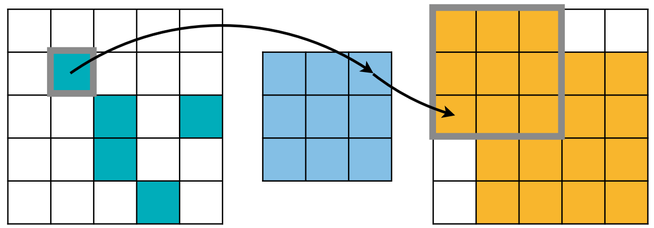
No compute
No compute
No compute
No compute
No compute
(P_0, Q_0, $W_{0,0}$)
No compute

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

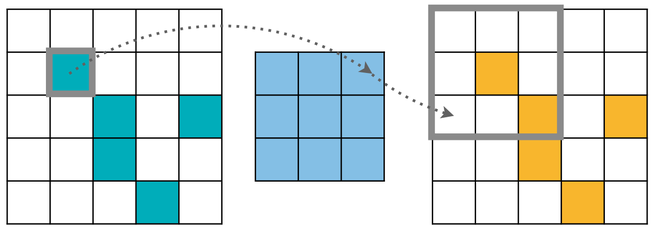
Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(**In**, **Out**, **Wgt**)

Computation
(**f_{Out} = f_{Out} + f_{In} × W_{Wgt}**) for each entry in the maps

(P_0, Q_0, $W_{1,1}$)
(P_0, Q_1, $W_{1,0}$)
(P_0, Q_2, $W_{1,-1}$)
(P_0, Q_3, $W_{0,1}$)
(P_0, Q_4, $W_{0,0}$)
(P_0, Q_5, $W_{0,-1}$)
(P_0, Q_8, $W_{-1,1}$)

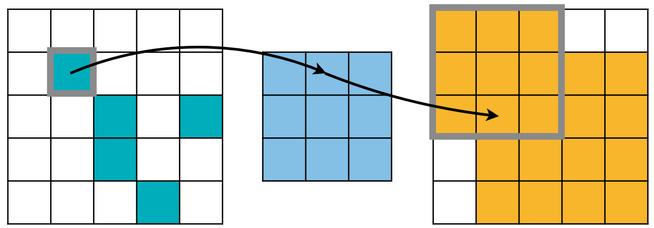
No compute
No compute
No compute
No compute
No compute
(P_0, Q_0, $W_{0,0}$)
No compute
No compute

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

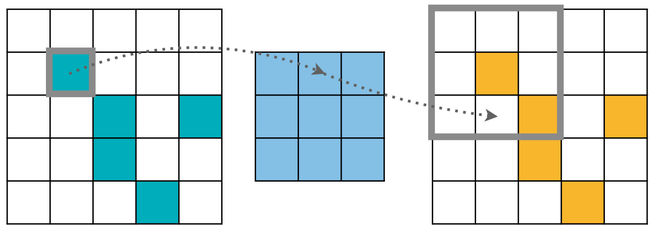
Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(**In**, **Out**, **Wgt**)

Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

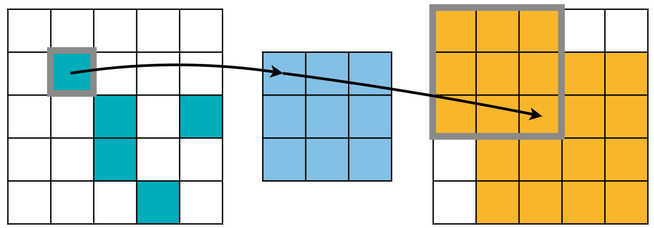
(P_0, Q_0, $W_{1,1}$)	No compute
(P_0, Q_1, $W_{1,0}$)	No compute
(P_0, Q_2, $W_{1,-1}$)	No compute
(P_0, Q_3, $W_{0,1}$)	No compute
(P_0, Q_4, $W_{0,0}$)	(P_0, Q_0, $W_{0,0}$)
(P_0, Q_5, $W_{0,-1}$)	No compute
(P_0, Q_8, $W_{-1,1}$)	No compute
(P_0, Q_9, $W_{-1,0}$)	No compute

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



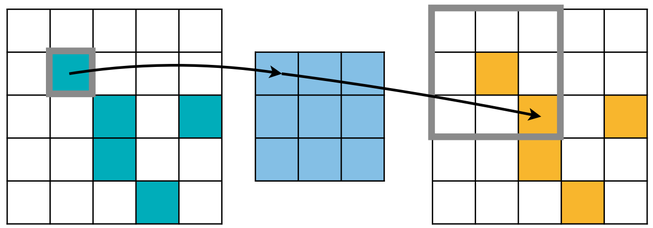
Maps
(In, Out, Wgt)

Computation
($f_{Out} = f_{Out} + f_{In} \times W_{Wgt}$) for each entry in the maps

(P_0, Q_0, $W_{1,1}$)
(P_0, Q_1, $W_{1,0}$)
(P_0, Q_2, $W_{1,-1}$)
(P_0, Q_3, $W_{0,1}$)
(P_0, Q_4, $W_{0,0}$)
(P_0, Q_5, $W_{0,-1}$)
(P_0, Q_8, $W_{-1,1}$)
(P_0, Q_9, $W_{-1,0}$)
(P_0, Q_{10}, $W_{-1,-1}$)

9 matrix multiplications

Sparse Convolution



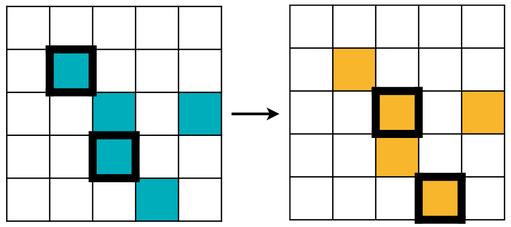
No compute
No compute
No compute
No compute
No compute
(P_0, Q_0, $W_{0,0}$)
No compute
No compute
No compute
No compute
(P_0, Q_1, $W_{-1,-1}$)

2 matrix multiplications

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Existing GPU implementation of sparse convolution

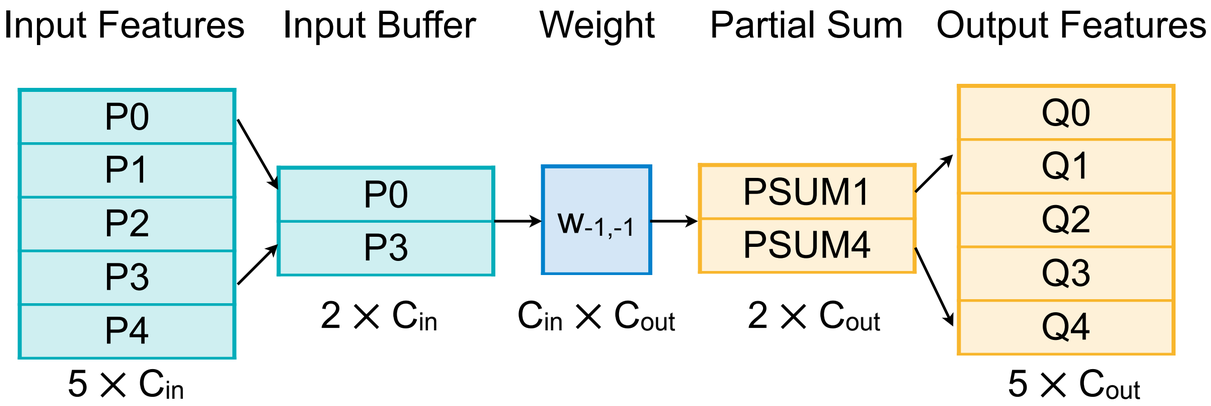
Weight-stationary computation, separate matmul for different weights



Workload

Maps
(In, Out, Wgt)

(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)
(P_1, Q_3, $W_{-1,0}$)
(P_0, Q_0, $W_{0,0}$)
(P_1, Q_1, $W_{0,0}$)
(P_2, Q_2, $W_{0,0}$)
(P_3, Q_3, $W_{0,0}$)
(P_4, Q_4, $W_{0,0}$)
(P_3, Q_1, $W_{1,0}$)
(P_1, Q_0, $W_{1,1}$)
(P_4, Q_3, $W_{1,1}$)



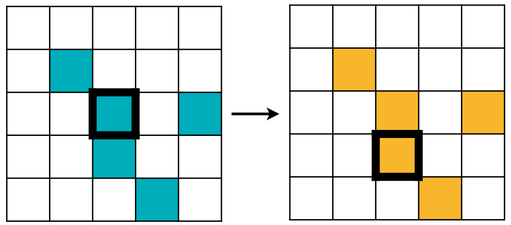
$f_1 = f_1 + f_0 \times W_{-1,-1}$

$f_4 = f_4 + f_3 \times W_{-1,-1}$

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Existing GPU implementation of sparse convolution

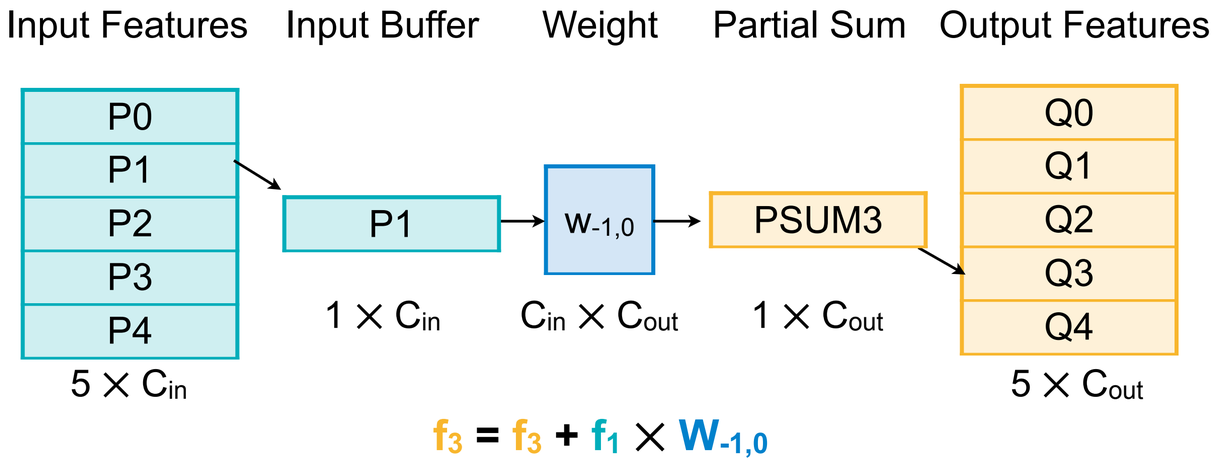
Weight-stationary computation, separate matmul for different weights



Workload

Maps
(In, Out, Wgt)

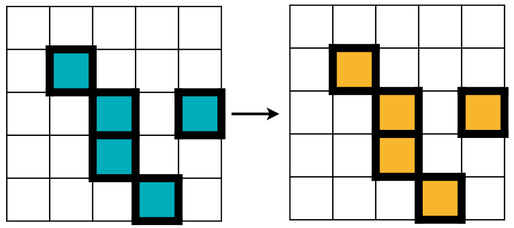
(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)
(P_1, Q_3, $W_{-1,0}$)
(P_0, Q_0, $W_{0,0}$)
(P_1, Q_1, $W_{0,0}$)
(P_2, Q_2, $W_{0,0}$)
(P_3, Q_3, $W_{0,0}$)
(P_4, Q_4, $W_{0,0}$)
(P_3, Q_1, $W_{1,0}$)
(P_1, Q_0, $W_{1,1}$)
(P_4, Q_3, $W_{1,1}$)



TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Existing GPU implementation of sparse convolution

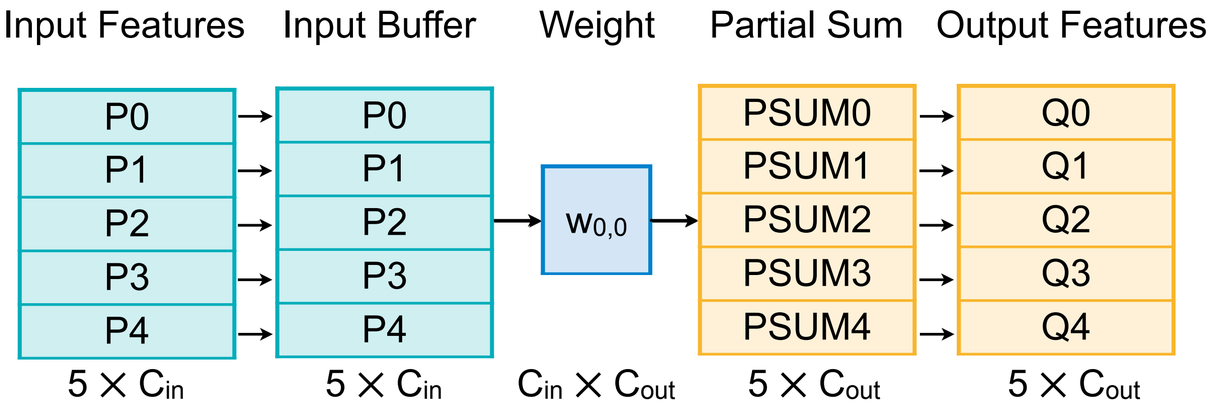
Weight-stationary computation, separate matmul for different weights



Workload

Maps
(In, Out, Wgt)

(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)
(P_1, Q_3, $W_{-1,0}$)
(P_0, Q_0, $W_{0,0}$)
(P_1, Q_1, $W_{0,0}$)
(P_2, Q_2, $W_{0,0}$)
(P_3, Q_3, $W_{0,0}$)
(P_4, Q_4, $W_{0,0}$)
(P_3, Q_1, $W_{1,0}$)
(P_1, Q_0, $W_{1,1}$)
(P_4, Q_3, $W_{1,1}$)



$$f_i = f_i + f_i \times W_{0,0}$$

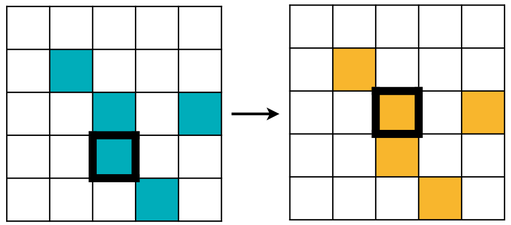
$$i = 0, 1, 2, 3, 4$$

Note: maps for $W_{0,0}$ contains all entries.

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Existing GPU implementation of sparse convolution

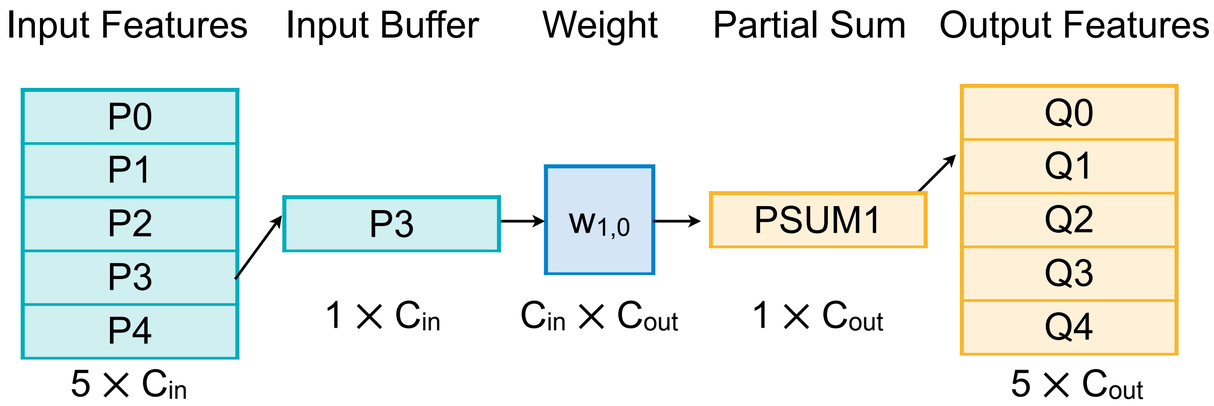
Weight-stationary computation, separate matmul for different weights



Workload

Maps
(In, Out, Wgt)

(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)
(P_1, Q_3, $W_{-1,0}$)
(P_0, Q_0, $W_{0,0}$)
(P_1, Q_1, $W_{0,0}$)
(P_2, Q_2, $W_{0,0}$)
(P_3, Q_3, $W_{0,0}$)
(P_4, Q_4, $W_{0,0}$)
(P_3, Q_1, $W_{1,0}$)
(P_1, Q_0, $W_{1,1}$)
(P_4, Q_3, $W_{1,1}$)

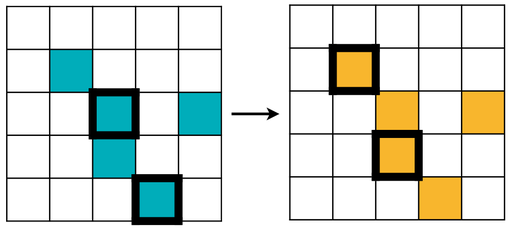


$f_1 = f_1 + f_3 \times W_{1,0}$

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Existing GPU implementation of sparse convolution

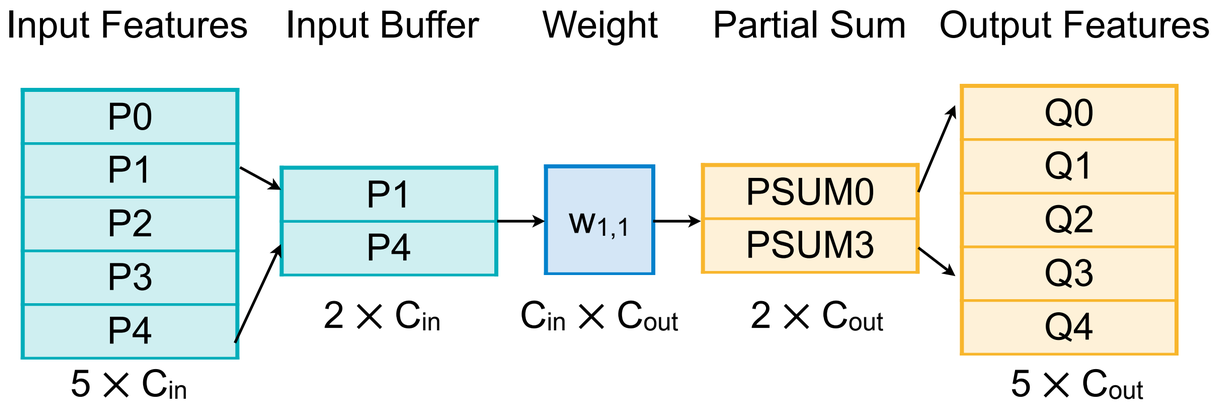
Weight-stationary computation, separate matmul for different weights



Workload

Maps
(In, Out, Wgt)

(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)
(P_1, Q_3, $W_{-1,0}$)
(P_0, Q_0, $W_{0,0}$)
(P_1, Q_1, $W_{0,0}$)
(P_2, Q_2, $W_{0,0}$)
(P_3, Q_3, $W_{0,0}$)
(P_4, Q_4, $W_{0,0}$)
(P_3, Q_1, $W_{1,0}$)
(P_1, Q_0, $W_{1,1}$)
(P_4, Q_3, $W_{1,1}$)

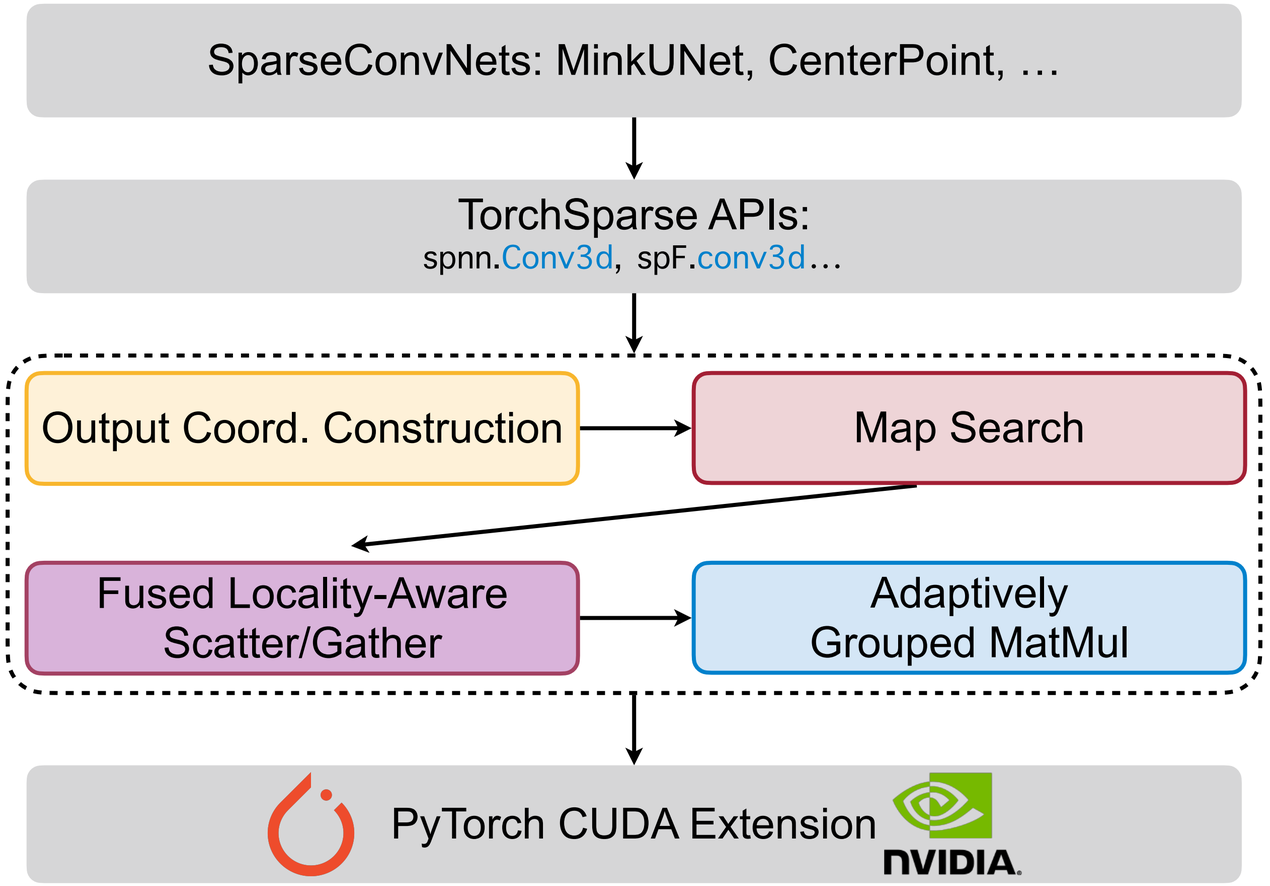


$f_0 = f_0 + f_1 \times W_{1,1}$

$f_3 = f_3 + f_4 \times W_{1,1}$

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

TorchSparse system overview



TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

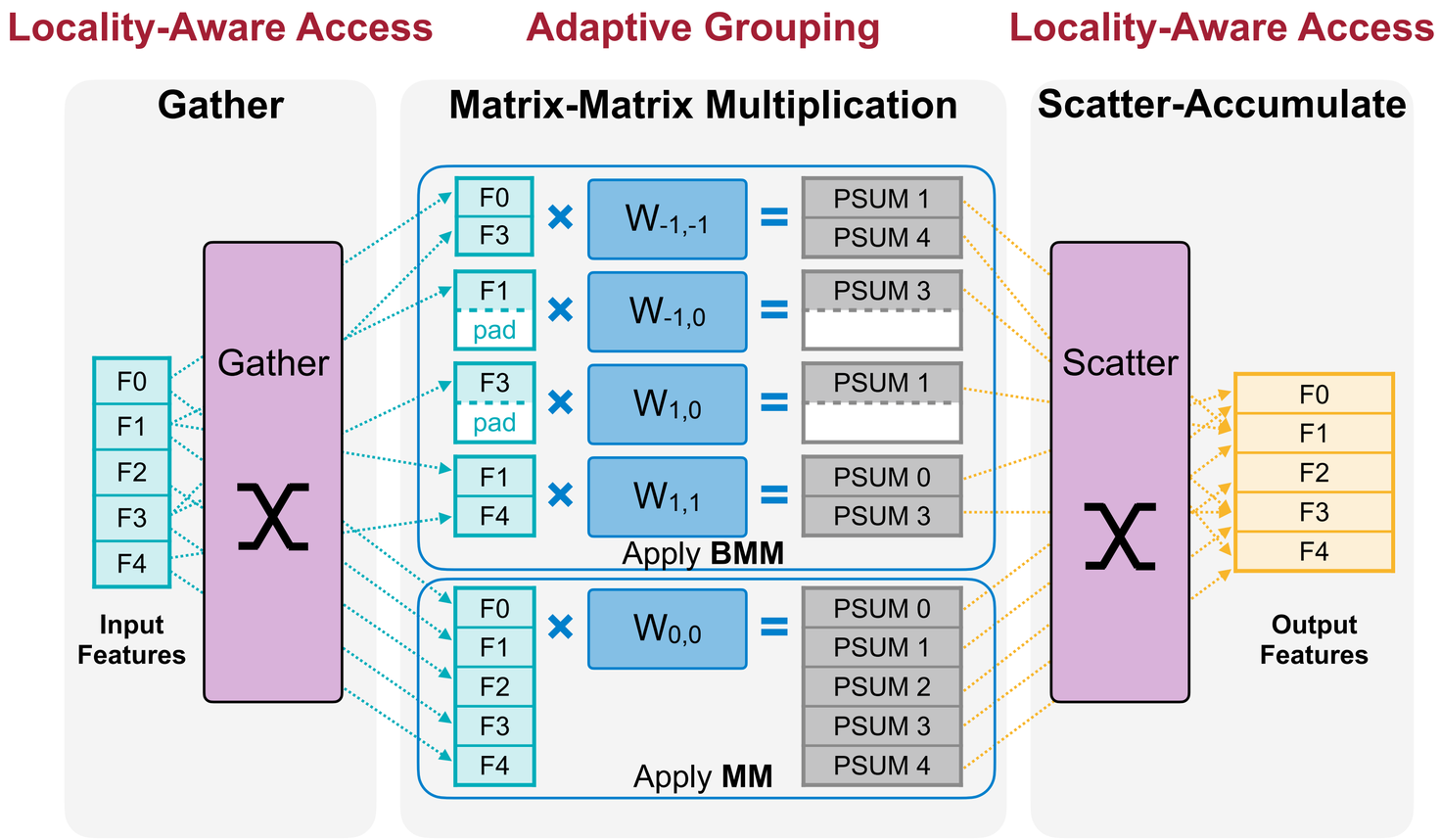
TorchSparse has PyTorch-like APIs

```python
import torch.nn as nn
class ConvBlock(nn.Sequential):
    def __init__(self,
                 in_channels: int,
                 out_channels: int,
                 kernel_size: Union[int, list, tuple],
                 stride: Union[int, list, tuple] = 1,
                 dilation: int = 1) -> None:
        super().__init__(
            nn.Conv2d(in_channels,
                      out_channels,
                      kernel_size,
                      stride=stride,
                      dilation=dilation),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(True)
        )
```

```python
import torchsparse.nn as spnn
class SparseConvBlock(nn.Sequential):
    def __init__(self,
                 in_channels: int,
                 out_channels: int,
                 kernel_size: Union[int, list, tuple],
                 stride: Union[int, list, tuple] = 1,
                 dilation: int = 1) -> None:
        super().__init__(
            spnn.Conv3d(in_channels,
                        out_channels,
                        kernel_size,
                        stride=stride,
                        dilation=dilation),
            spnn.BatchNorm(out_channels),
            spnn.ReLU(True)
        )
```

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

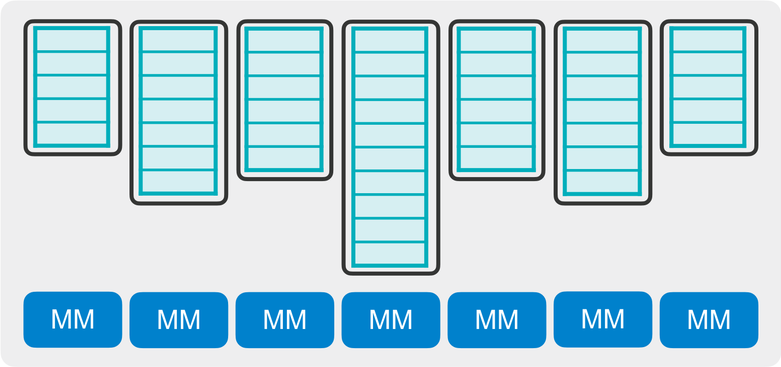
TorchSparse optimization overview



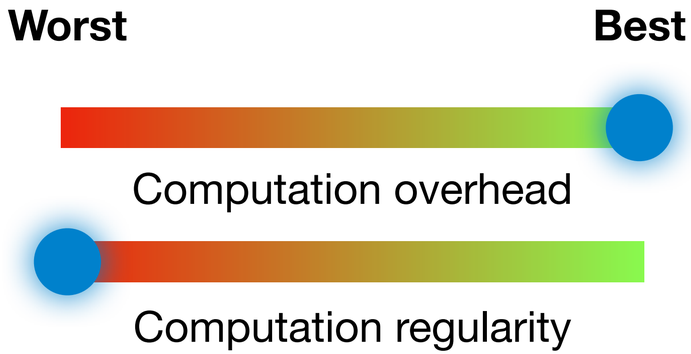
TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Trading computation for regularity

Separate computation (baseline) : many kernel calls, low device utilization



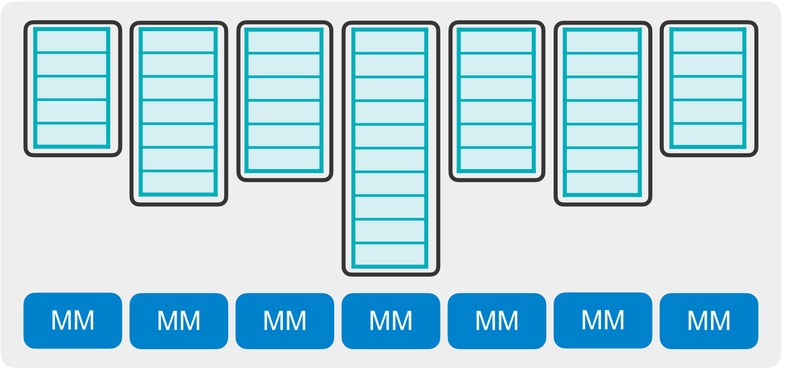
Separate Computation



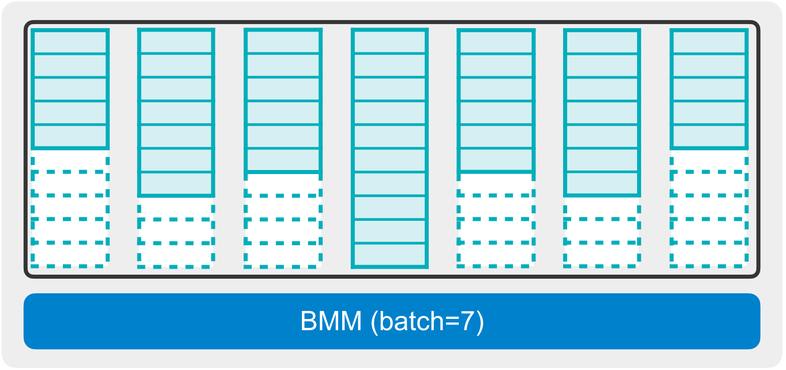
TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Trading computation for regularity

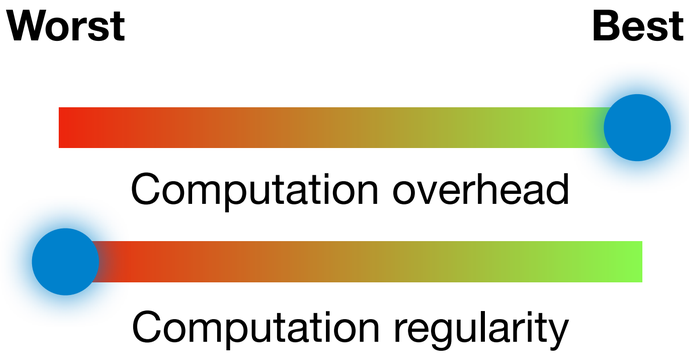
Dense convolution: best regularity but large computation overhead

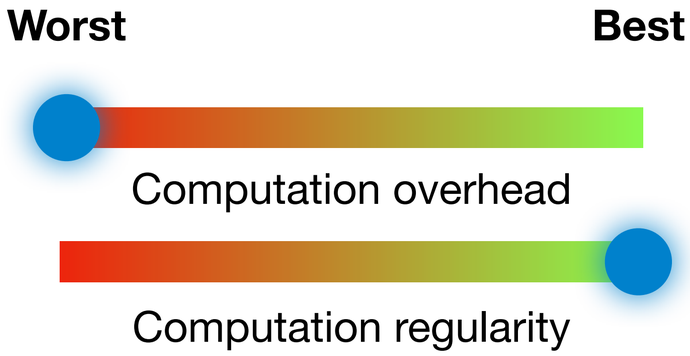


Separate Computation



Dense Convolution

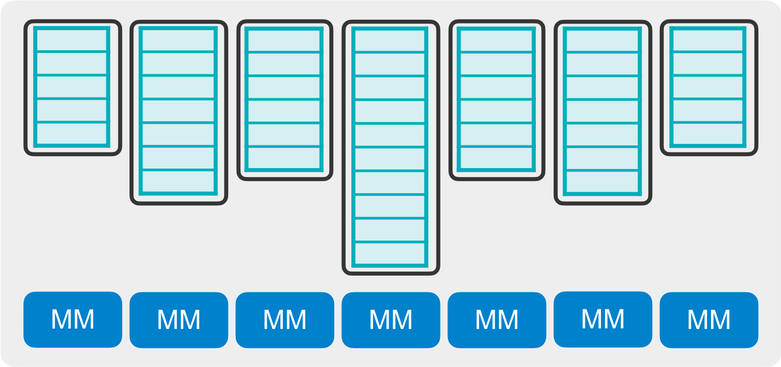




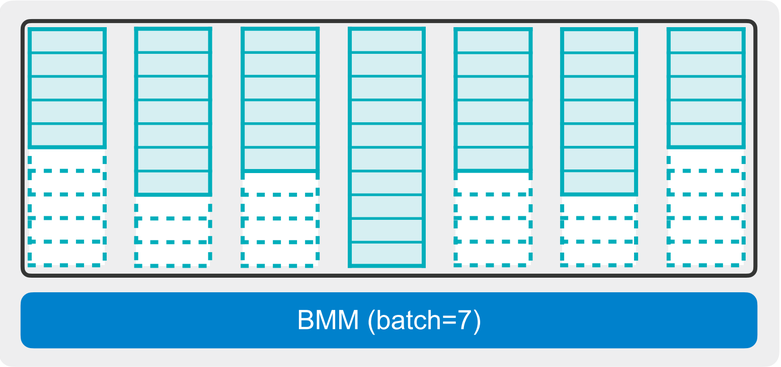
TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Trading computation for regularity

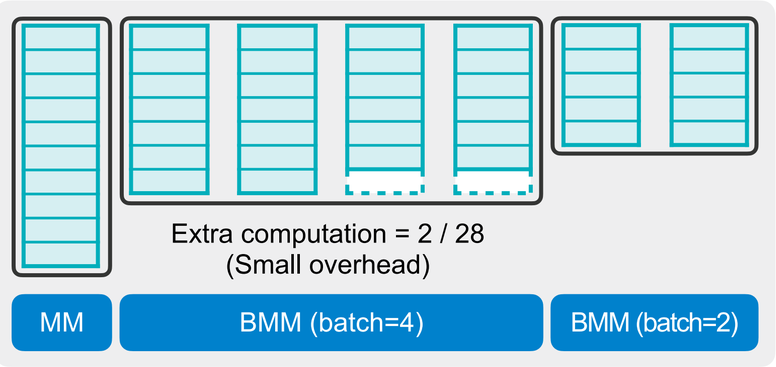
Computation with grouping: balancing overhead and regularity



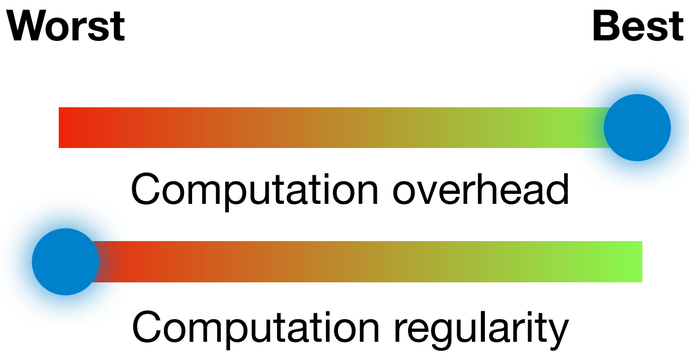
Separate Computation

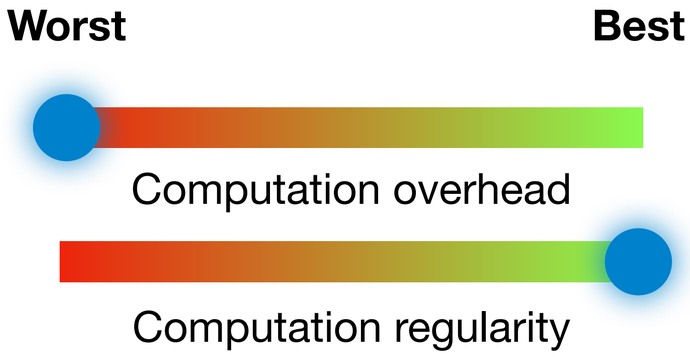


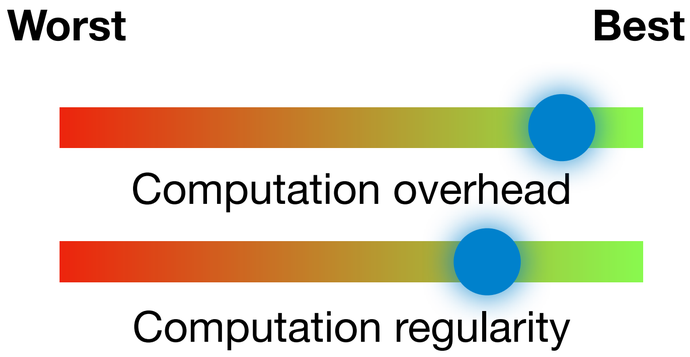
Dense Convolution



Computation with grouping



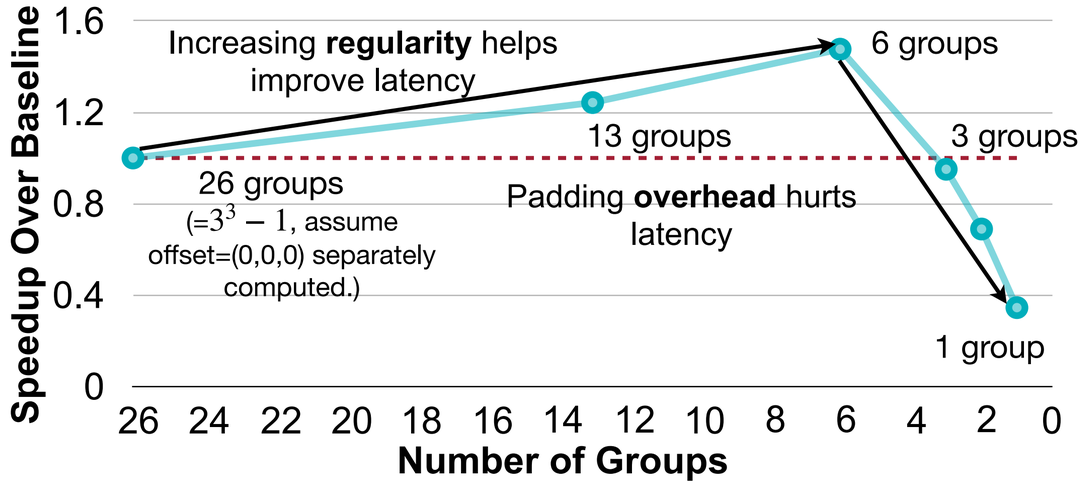


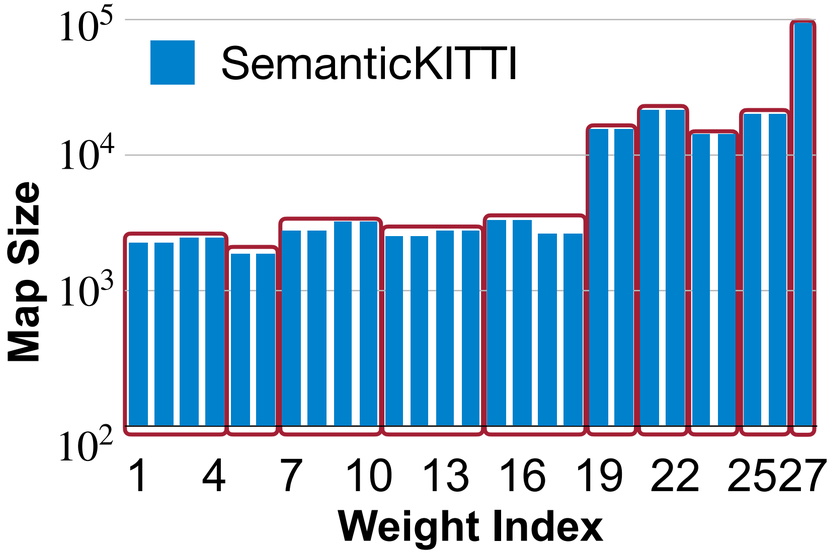


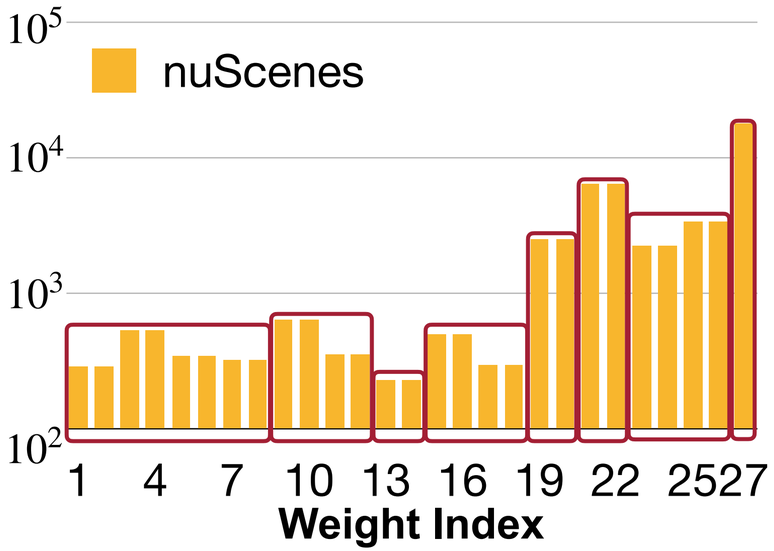
TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Trading computation for regularity

Searching customized strategy for different model and datasets

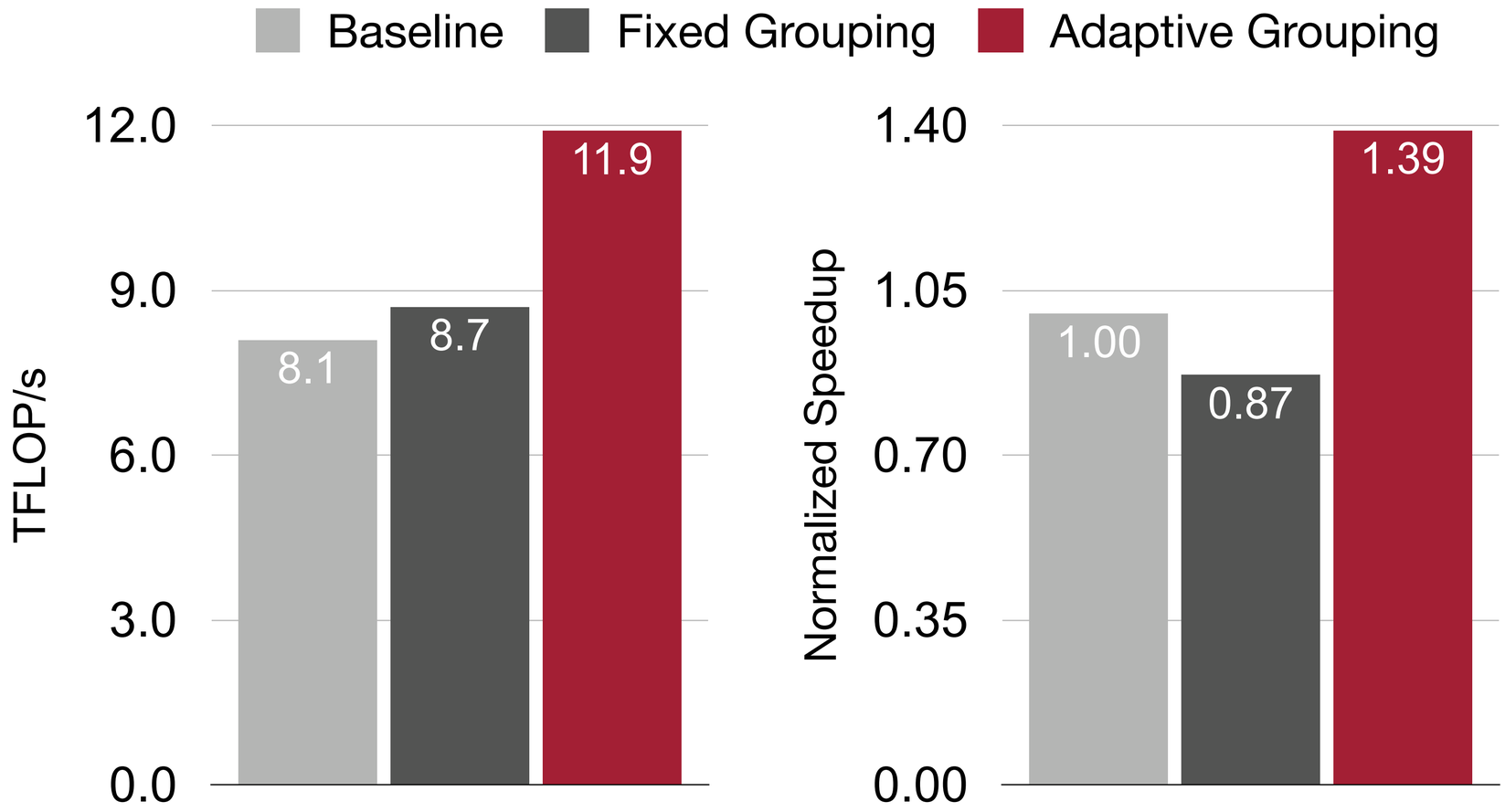






Results on matrix multiplication optimizations

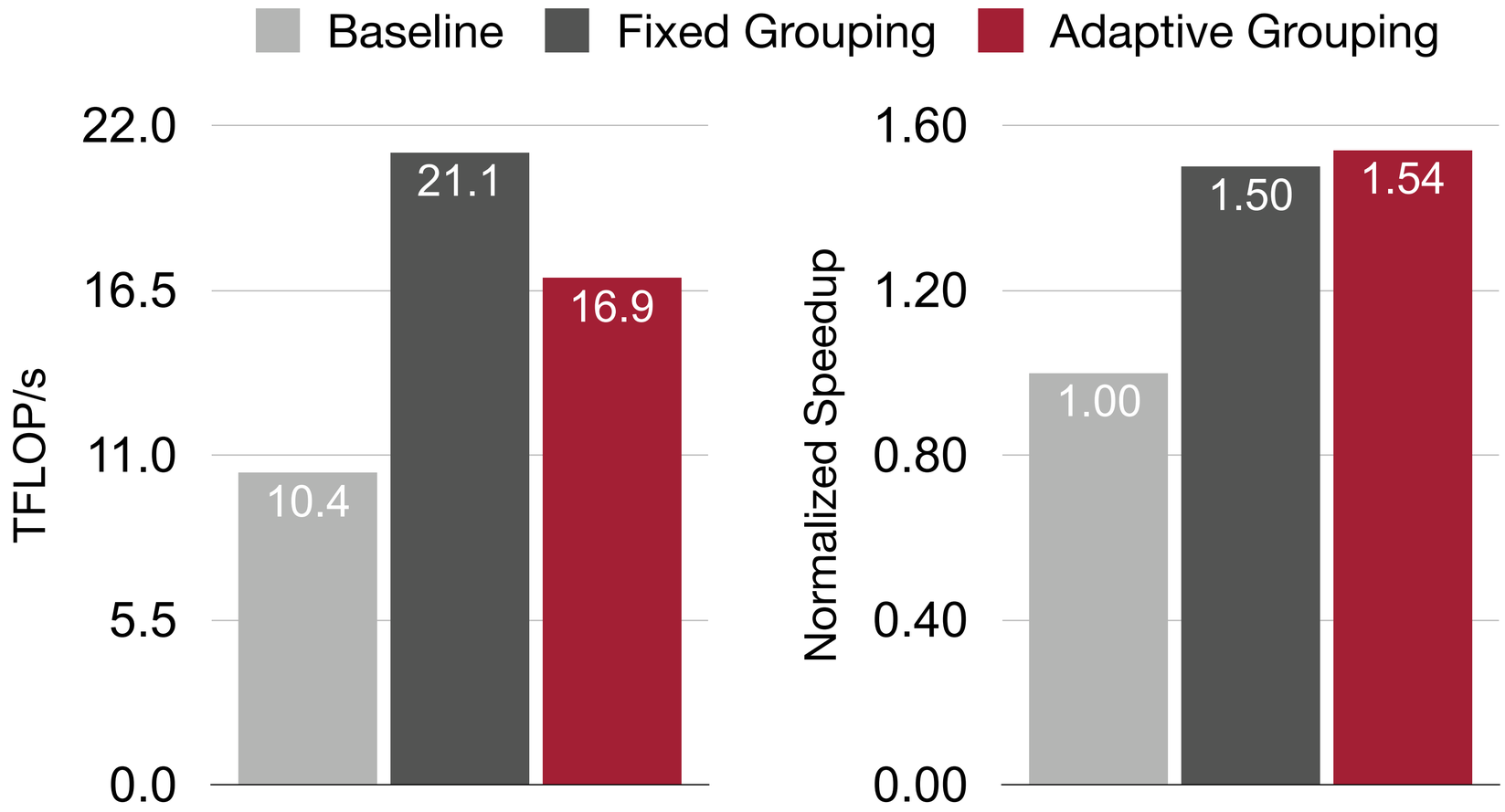
SemanticKITTI



TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Results on matrix multiplication optimizations

nuScenes: fixed grouping has best TFLOP/s but adaptive grouping is faster

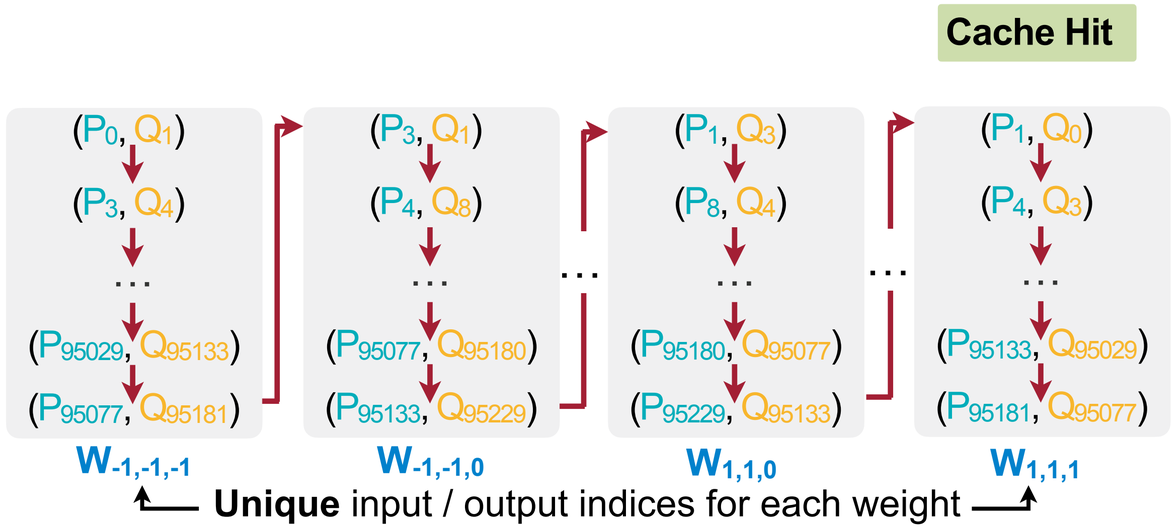


This is because fixed grouping introduced large amount of **redundant computation**.

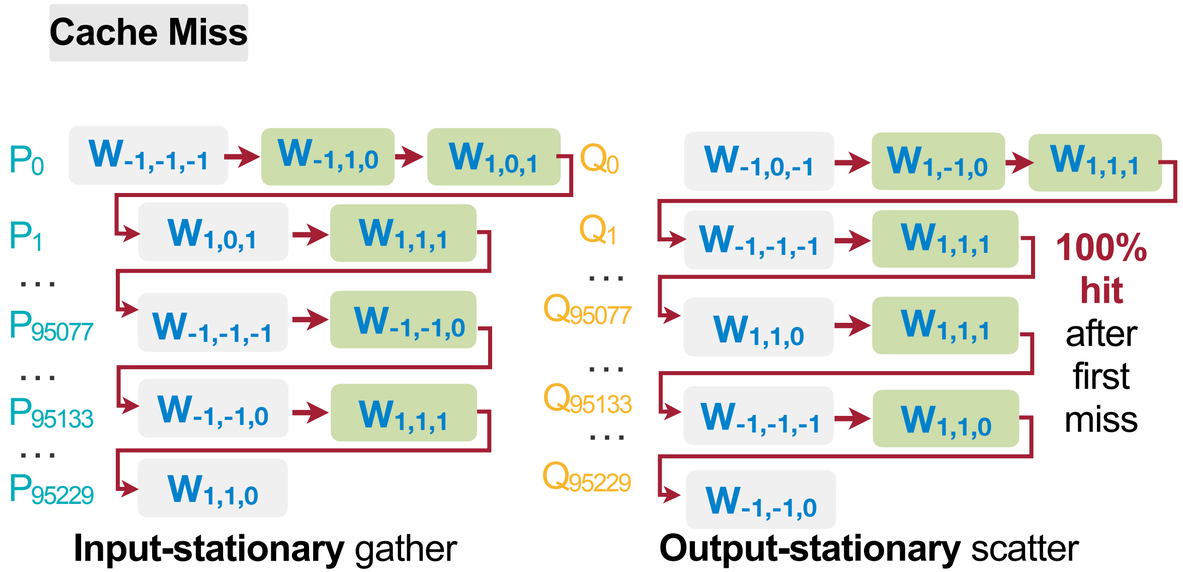
TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Solution: fused and locality-aware scatter-gather

Improving the cache hit ratio via reordering memory accesses



(a) baseline: weight-stationary scatter-gather **weight-stationary**

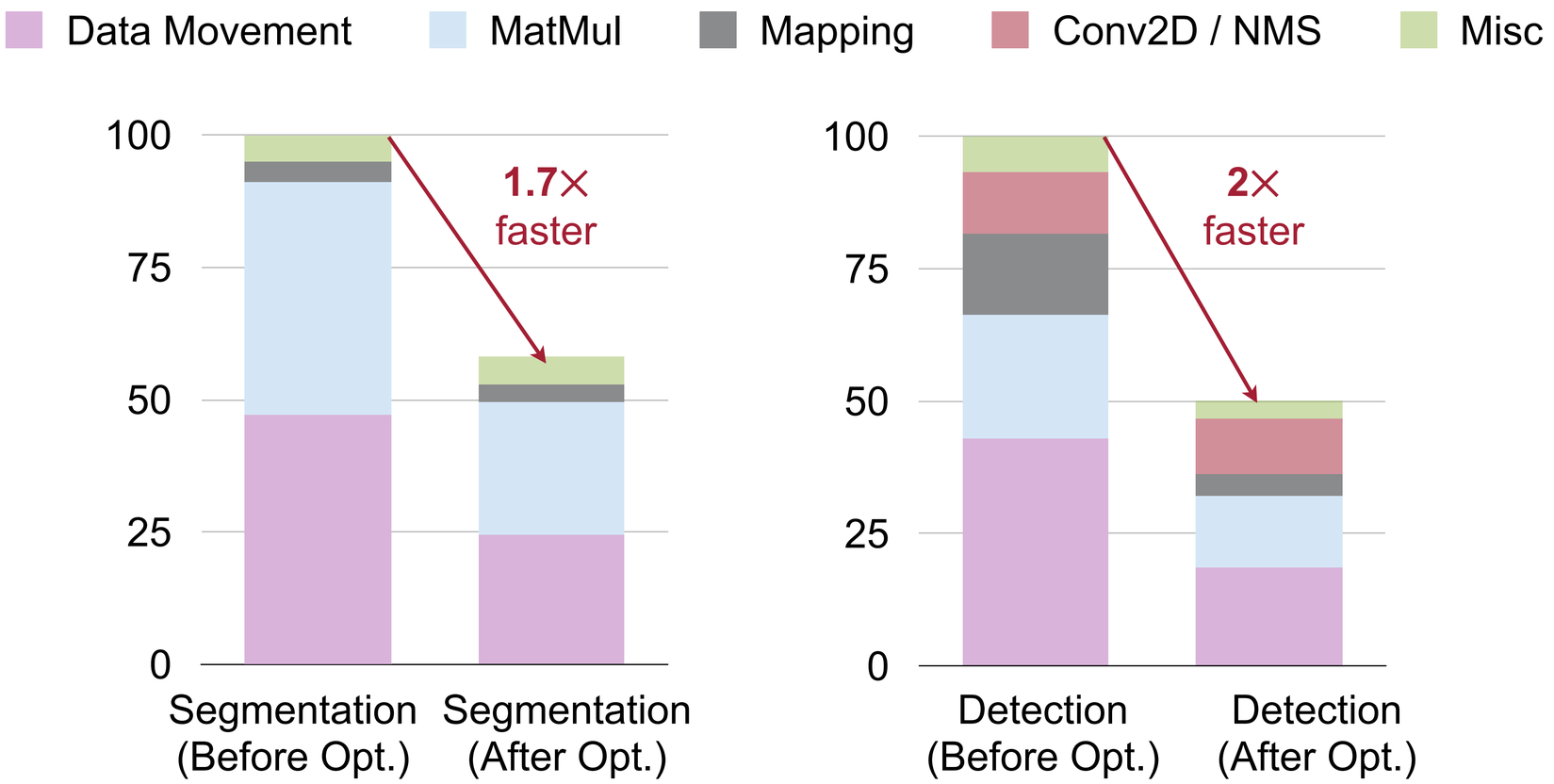


(b) ours: fused and locality-aware scatter-gather

TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

Results for TorchSparse Optimizations

Trade computation for regularity and reduce memory footprint

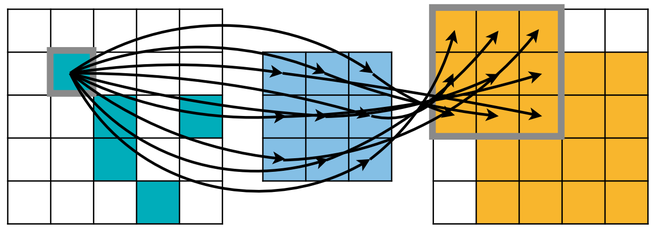


TorchSparse: Efficient Point Cloud Inference Engine [Tang *et al.*, MLSys 2022]

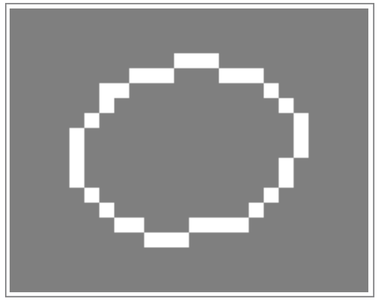
3. Hardware Support

Efficient Point Cloud Recognition and Perception

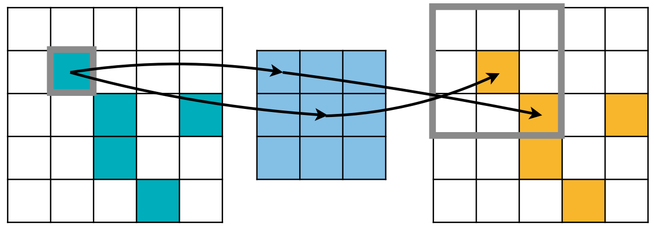
Accelerating Point Cloud is Challenging



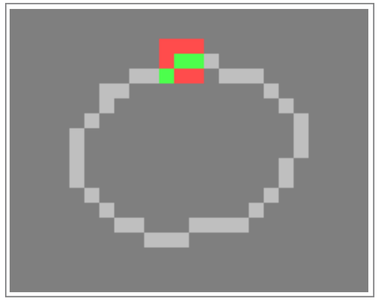
Each nonzero input is multiplied with all nonzero weights



Previous Accelerators: SCNN, Cambricon-X, Cnvlutin, *etc*

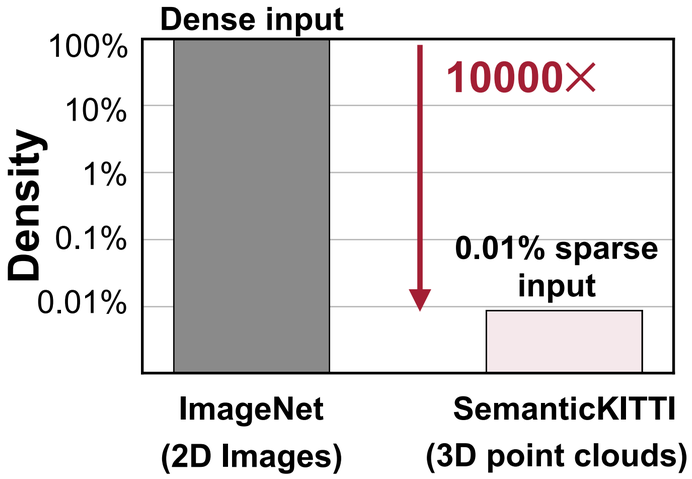


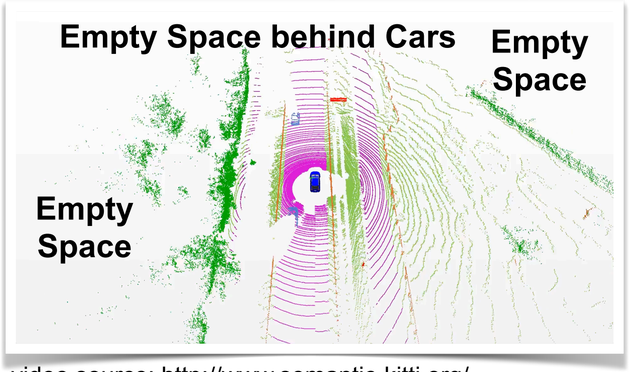
Each nonzero input is not multiplied with all nonzero weights



Previous Accelerators: **None!**

- Plenty of accelerators for 2D vision with dense inputs, but very *limited* research investigated **3D vision with sparse inputs** (point clouds).
- Point clouds are spatially sparse, which is fundamentally different from the weight sparsity or activation sparsity commonly seen in the 2D vision.

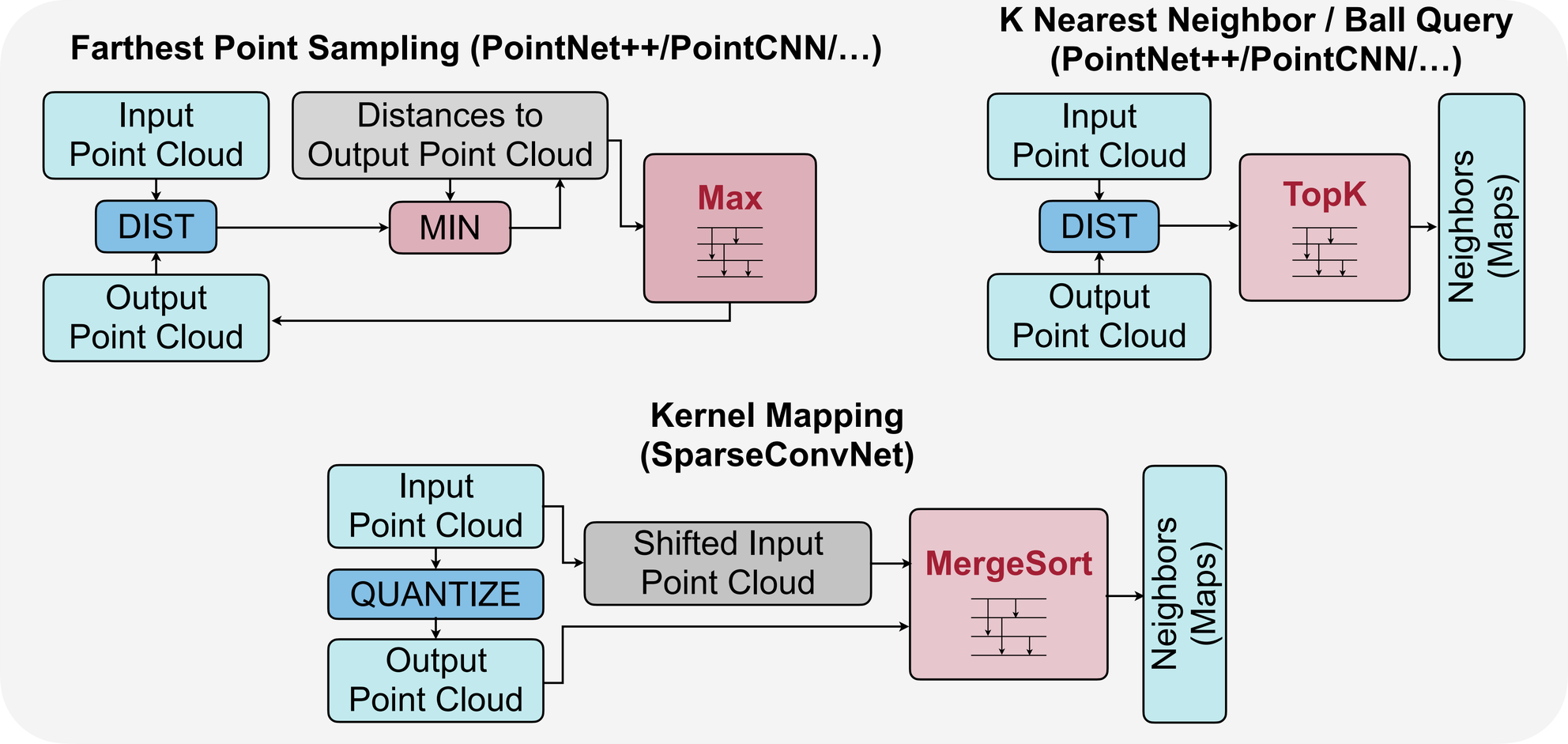




video source: http://www.semantic-kitti.org/

Mapping Unit

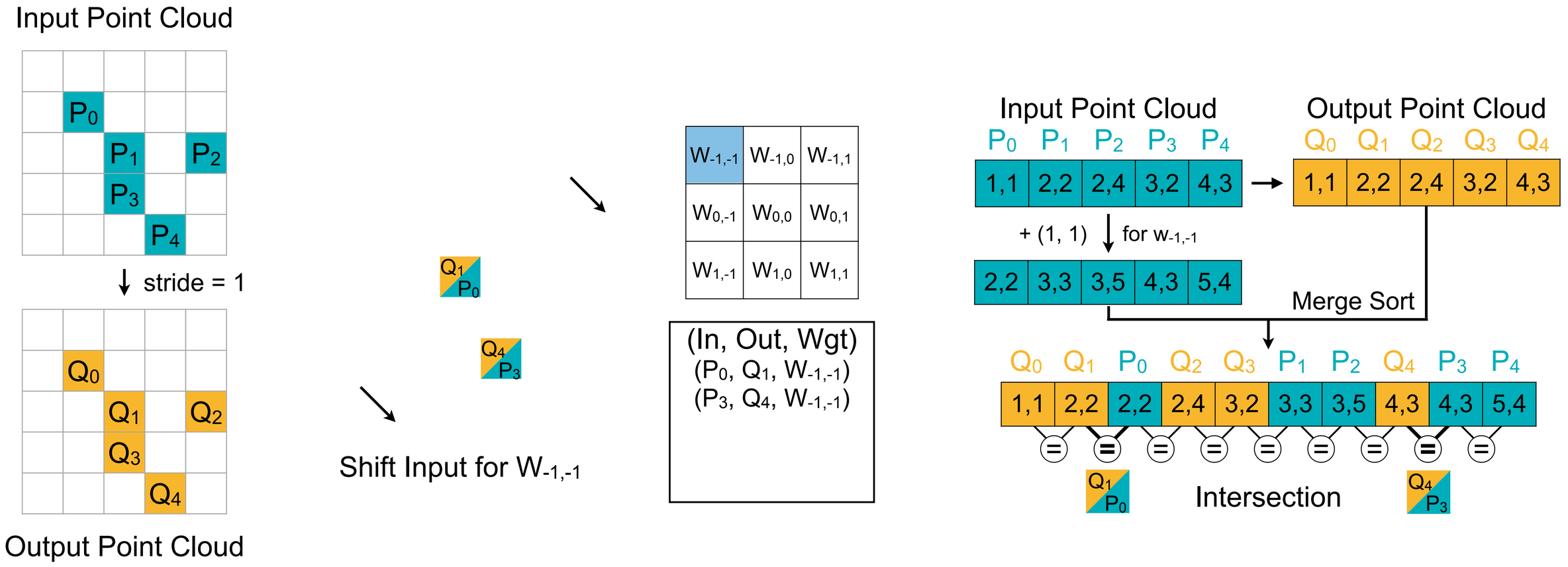
Diverse Mapping Operations in One Versatile Architecture



PointAcc: Efficient Point Cloud Accelerator [Lin *et al*., MICRO 2021]

Mapping Unit

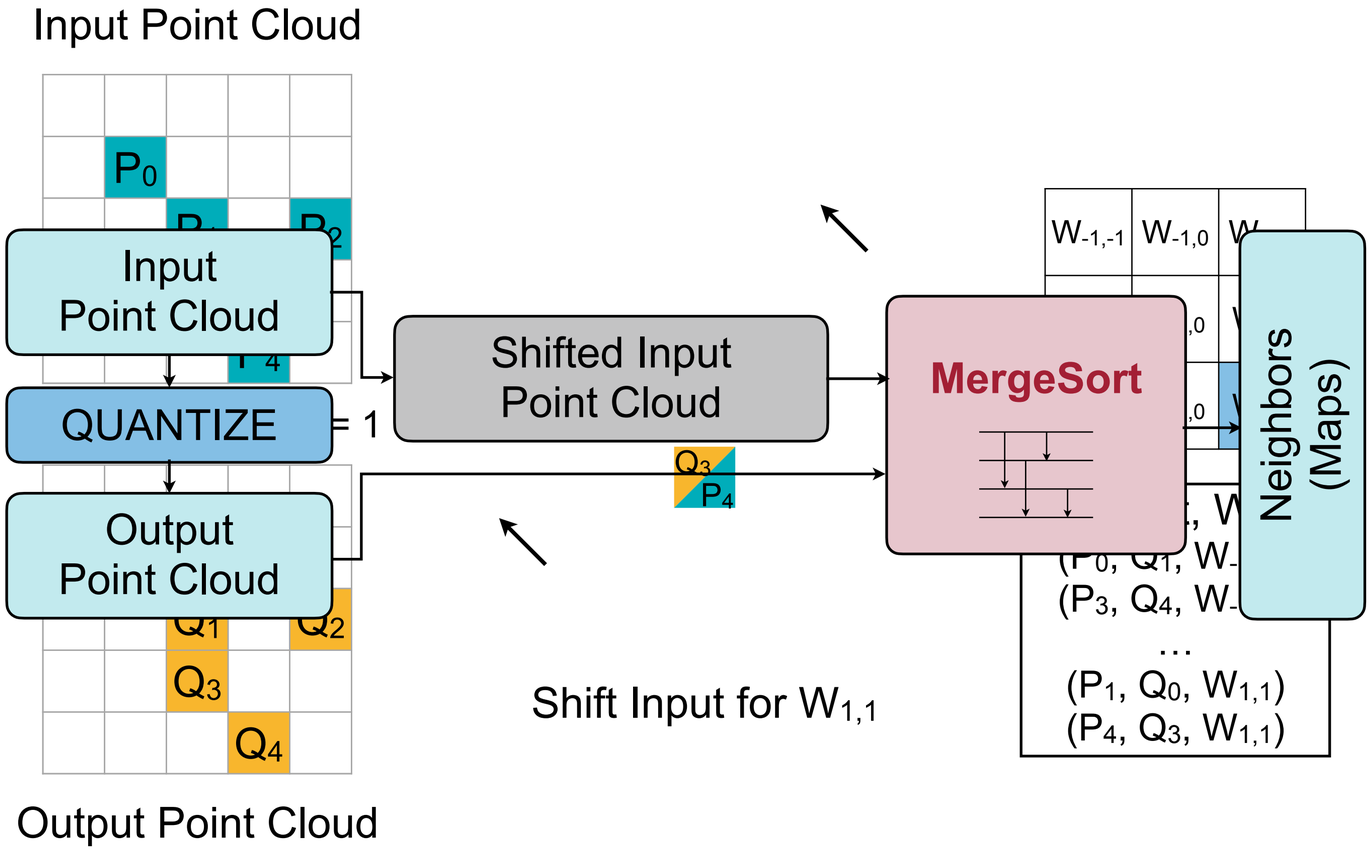
Merge sort can be used to find mappings in sparse convolution

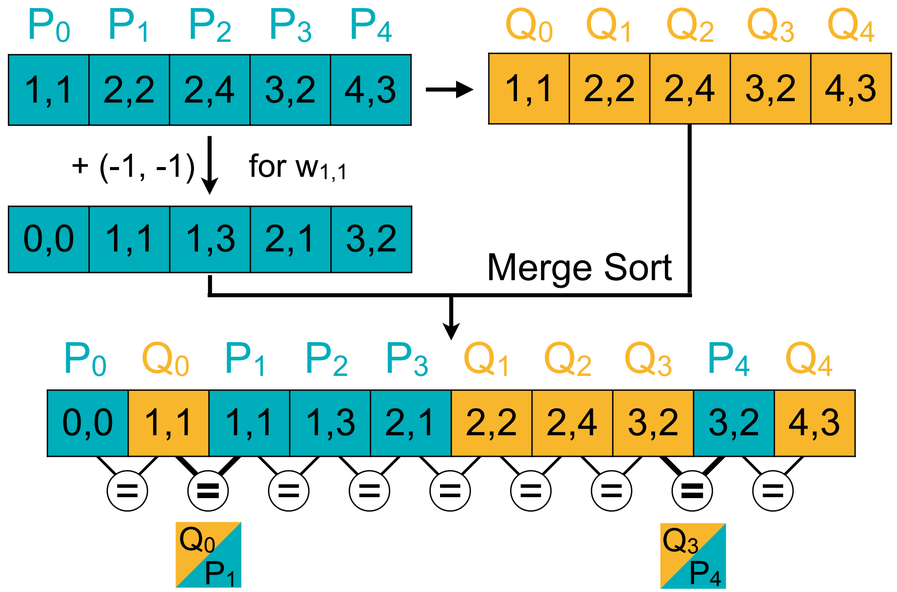


PointAcc: Efficient Point Cloud Accelerator [Lin *et al.*, MICRO 2021]

Mapping Unit

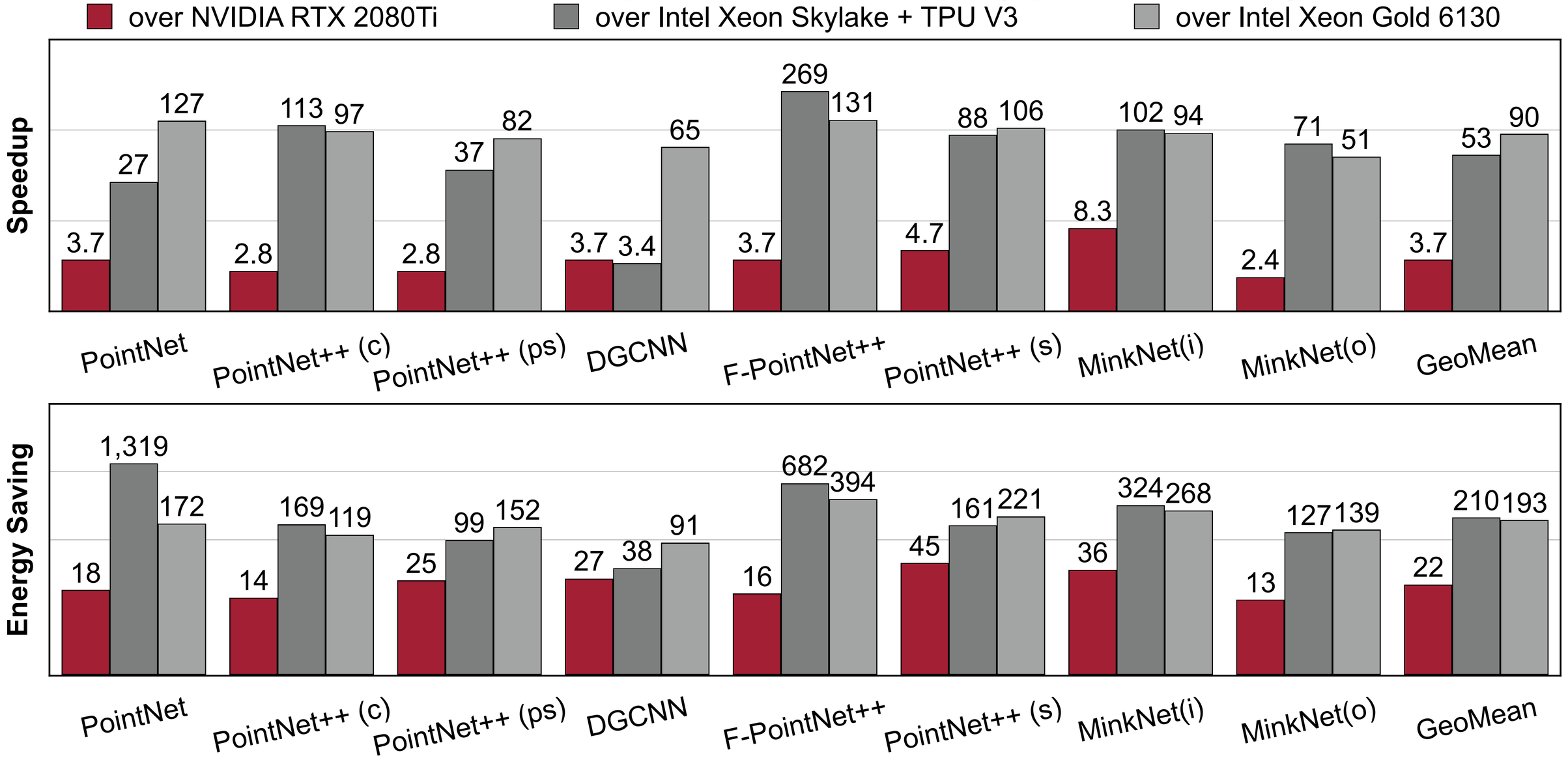
Merge sort can be used to find mappings in sparse convolution





PointAcc: Efficient Point Cloud Accelerator [Lin *et al.*, MICRO 2021]

PointAcc: Speedup and Energy Saving



PointAcc: Efficient Point Cloud Accelerator [Lin *et al.*, MICRO 2021]

Full-Stack Design for Point Cloud Learning

Algorithm-Software-Hardware Co-Design and Efficient Acceleration

Full-stack optimizations

Algorithm

3D Light-weight Neural-net [PVCNN, NeurIPS'19 Spotlight]

3D Neural Architecture Search [SPVNAS, ECCV'20]

Software

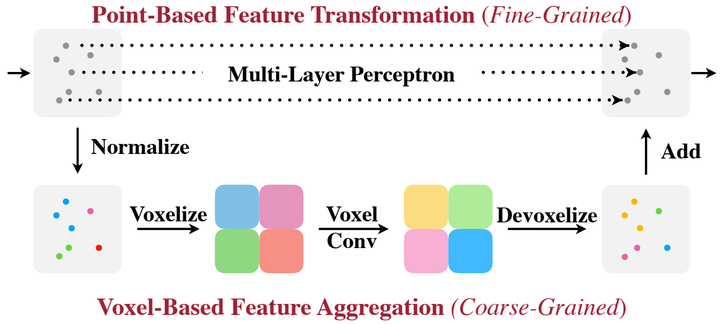
3D Inference Engine on GPU

[TorchSparse, MLSys'22]

Hardware

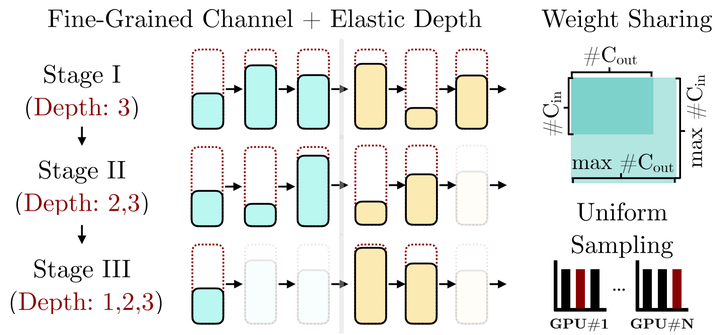
3D Hardware Accelerator

[PointAcc, Micro'21]



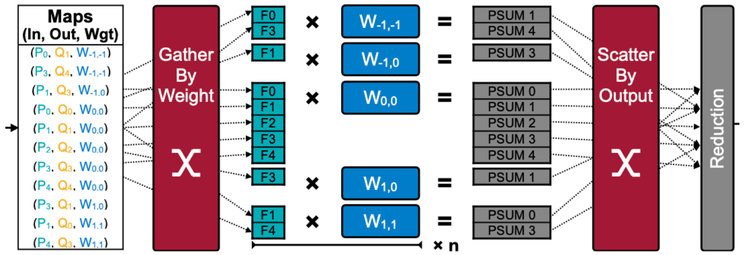
[PVCNN, NeurIPS'19 Spotlight]

New design space, new primitive for point cloud



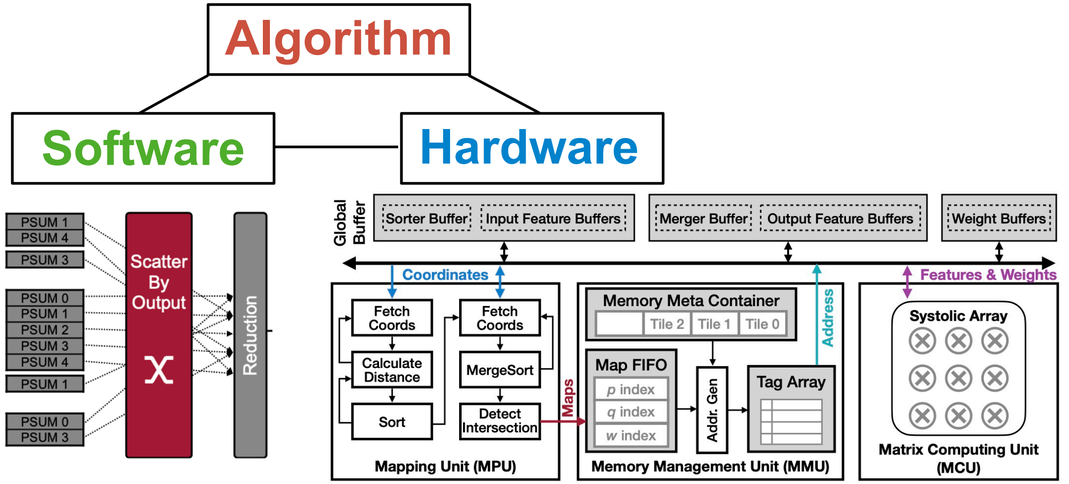
[SPVNAS, ECCV'20]

3D neural architecture search



[TorchSparse, MLSys'22]

GPU library for 3D sparse convolution

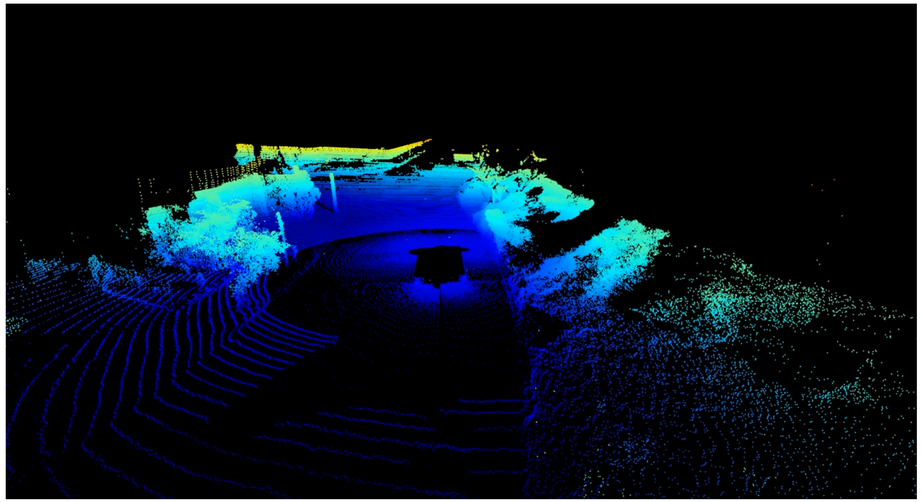


[PointAcc, Micro'21]

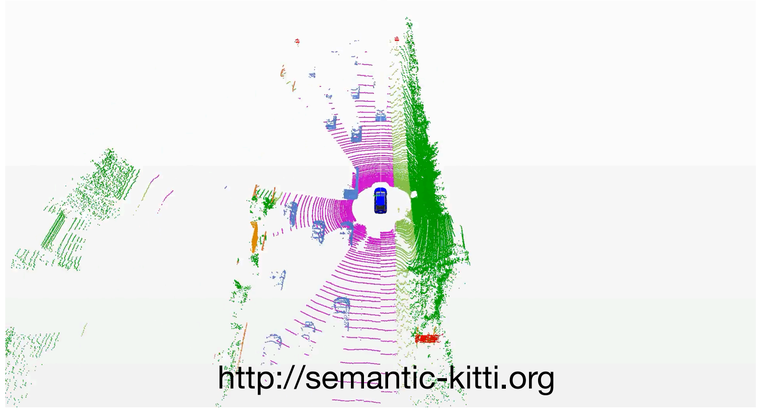
Hardware accelerator for point cloud

Summary of Today's Lecture

- **In this lecture, we introduce:**
- 3D point clouds and 3D data representation: multi-view images, voxels, point sets, sparse voxels and octaves.
- Efficient algorithms for deep learning on 3D data.
- Efficient systems and specialized hardware for point cloud processing.
- **In the next lecture, we will introduce:**
- Efficient video understanding and GANs.

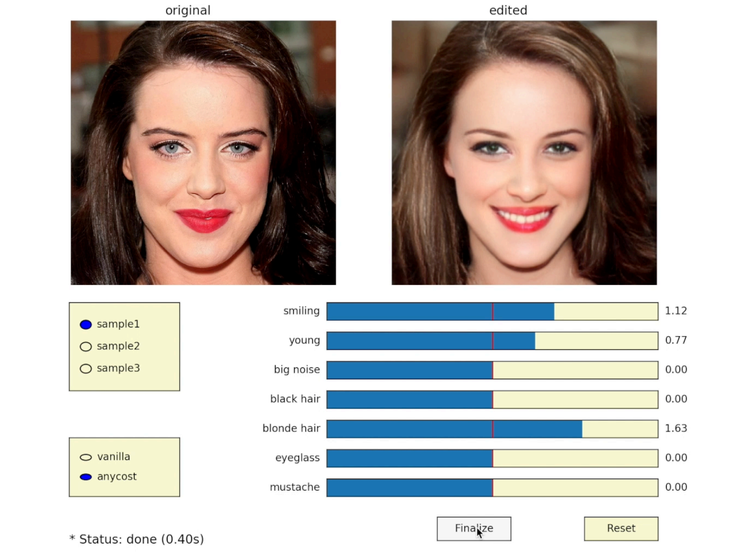


Application-Specific Optimizations for Efficient AI Computing

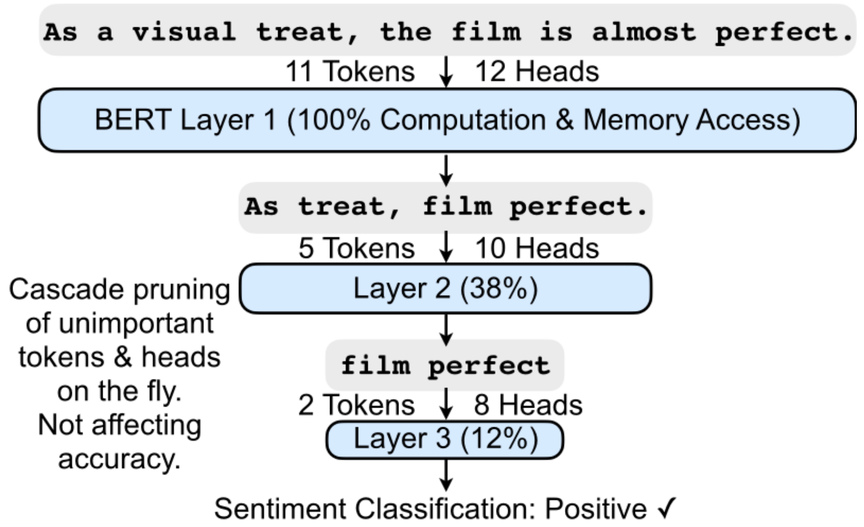


Point cloud: 3D spatial redundancy (this lecture)

Videos: temporal redundancy



GANs: 2D spatial redundancy



NLP: token redundancy

References

- 1. 3D Semantic Parsing of Large-Scale Indoor Spaces [Armeni et al., CVPR 2016]
- 2. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds [Hu et al., CVPR 2020]
- 3. Vision meets robotics: The KITTI dataset [Geiger et al., IJRR 2013]
- 4. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks [Choy et al., CVPR 2019]
- 5. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang et al., ECCV 2020]
- 6. Volumetric and Multi-View CNNs for Object Classification on 3D Data [Qi *et al.*, CVPR 2016]
- 7. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud [Wu et al., ICRA 2018]
- 8. SalsaNext: Fast, Uncertainty-aware Semantic Segmentation of LiDAR Point Clouds for Autonomous Driving [Cortinhal et al., arXiv 2020]
- 9. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation [Milioto et al., IROS 2019]
- 10. PointPillars: Fast Encoders for Object Detection from Point Clouds [Lang et al., CVPR 2019]

References

- 11. PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation [Zhou et al., CVPR 2020]
- 12. How to Voxelize Meshes and Point Clouds in Python [Nikolov, 2022]
- 13. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition [Maturana and Scherer, IROS 2015]
- 14. Point-Voxel CNN for Efficient 3D Deep Learning [Liu *et al.*, NeurIPS 2019]
- 15. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation [Qi *et al.*, CVPR 2017
- 16. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space [Qi et al., NeurIPS 2017]
- 17. Dynamic Graph CNN for Learning on Point Clouds [Wang et al., SIGGRAPH 2019]
- 18. PointCNN [Li *et al.*, NeurIPS 2018]
- 19. PointConv: Deep Convolutional Networks on 3D Point Clouds [Wu et al., CVPR 2019]
- 20. KPConv: Flexible and Deformable Convolution for Point Clouds [Thomas et al., ICCV 2019]

References

- 21. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds [Hu et al., CVPR 2020]
- 22. Submanifold Sparse Convolutional Neural Networks [Graham, BMVC 2015]
- 23. TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]
- 24. https://en.wikipedia.org/wiki/Octree
- 25. OctNet: Learning Deep 3D Representations at High Resolutions [Riegler et al., CVPR 2017]
- 26. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis [Wang et al., ToG 2018]
- 27. RPVNet: A Deep and Efficient Range-Point-Voxel Fusion Network for LiDAR Point Cloud Segmentation [Xu et al., ICCV 2021]
- 28. PointAcc: Efficient Point Cloud Accelerator [Lin *et al*., MICRO 2021]