Supplementary Material - Poly-PC: A Polyhedral Network for Multiple Point Cloud Tasks at Once

Tao Xie[1,3] Shiguang Wang[2,3] Ke Wang[1] Linqi Yang[1] Zhiqiang Jiang[1]
Xingcheng Zhang[3] Kun Dai[1] Ruifeng Li[1*] Jian Cheng[2]
[1]Harbin Institute of Technology [2]University of Electronic Science and Technology of China [3]SenseTime Research
{xietao1997, wangke, nana07, jzq, daikun, lrf100}@hit.edu.cn
{wangshiguang, chenjian}@uestc.edu.cn {zhangxingcheng}@sensetime.com

A. The visualization of task prioritization

In this part, we give the visualization of task prioritization when Poly-PC jointly optimizes three tasks in the main paper: 3D point shape classification, segmentation, and object detection. As shown in Fig. 1, the task prioritization of such three tasks adjusts dynamically and tends to converge with the progress of iteration. The task prioritization can enable our proposed gradient balance algorithm to prioritize the learning of difficult task at each epoch and ensures that all tasks converge to the optimal solution.

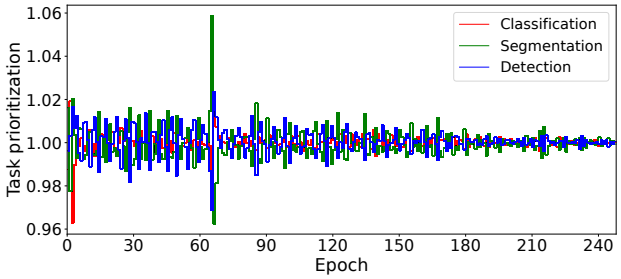


Figure 1. Illustration of task prioritization trend.

B. Search space of Poly-PC (large)

We design a large search space that includes five variable factors in Res-SA building layers: neighbour points, group radius, Res-MLP layer numbers in Φ_{mid} and Φ_{post}, reduction rate ϵ_1 in Φ_{mid}, expansion rate ϵ_2 in Φ_{post}, and output channel number in Res-SA layer. We also search the channel of stem MLP. We make the layer numbers of Φ_{mid} and Φ_{post} equal to reduce the whole search space. Note that each task has its own search space. We give the search space of Poly-PC (base) in Tab. 1 and Poly-PC (large) in Tab. 2.

*Corresponding author.

C. Search pipline

C.1. Supernet training

During supernet training, Poly-PC simultaneously learns K different tasks, that is, finding K mappings from K different datasets $\{\chi_k\}$ to a task-specific set of labels $\{y_k\}$, $k = 1, 2, ..., K$. Note that since each task has its own dataset domain, we need to use multiple GPUs to optimize the tasks, where the task-shared parameters are optimized in the global group while the task-specific parameters are optimized in the task-specific group, which is implemented by defining multiple communication groups in DistributedDataParallel of PyTorch. Specifically, in each training iteration, we first randomly sample one architecture α_k for task k. Then we retrieve its weights W_k from the supernet's weights and compute loss based on the sampled subnet, where the intersected parts of K tasks are considered task-shared weights and the others are viewed as task-specific weights. At last, we update the task-shared weights in the global group and the task-specific weights in task-specific group. In such process, no gradient or weight update acts on unsampled parameters in the supernet. We train Poly-PC for 50k iterations total with 24 Tesla V100 GPUs, that is, classification, segmentation and detection takes up 8 GPUs respectively. We view each 200 iterations as an epoch.

Implementation of parameter sharing. We can obtain the gradient vector $g_1^*, g_2^*, g_{chard}, g_K^*$ that are distributed on different GPUs from Eq. (9) in the main paper. Then, we get the final gradient vector of the shared parameters $g_{\theta_s}^*$ by All-Reduce operation (PyTorch) across all GPUs (also Eq. (10) in the main paper). In this way, the gradient vectors of shared parameters on all GPUs is $g_{\theta_s}^*$. If we ensure that the initialization of parameters, learning rate and weight decay for each task are the same, we can finally get task-shared parameters for all tasks, even in a single layer, we can still obtain the partially shared and unshared parameters. For task-specific parameters, we update the parameters directly

		Stem Channel	Neigh Num	Group Radius	Res-MLP Num	Output Channel	Expansion rate	Reduction rate
stage1	cls	(64, 72, 80)	(32, 40, 48)	(0.09, 0.1, 0.11)	(0, 1)	(128, 136, 144)	(4, 5)	(0.5, 0.75)
	det	(64, 72, 80)	(56, 64, 72)	(0.18, 0.2, 0.22)	(0, 1)	(128, 136, 144)	(4, 5)	(0.5, 0.75)
	seg	(32, 40, 48)	(40, 44, 48)	(0.09, 0.1, 0.11)	(0, 1)	(64, 72, 80)	(4, 5)	(0.5, 0.75)
stage2	cls	-	(32, 40, 48)	(0.18, 0.2, 0.22)	(0, 1)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
	det		(28, 32, 36)	(0.36, 0.4, 0.44)	(0, 1)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
	seg		(40, 44, 48)	(0.18, 0.2, 0.22)	(0, 1)	(128, 136, 144)	(4, 5)	(0.5, 0.75)
stage3	cls	-	(32, 40, 48)	(0.36 ,0.4, 0.44)	(0, 1)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
	det		(12, 16, 20)	(0.72, 0.8, 0.88)	(0, 1)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
	seg		(40, 44, 48)	(0.36, 0.4, 0.44)	(0, 1)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
stage4	cls	-	(32, 40, 48)	(0.36, 0.4, 0.44)	(0, 1)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
	det		(6, 8, 10)	(1.04, 1.2, 1.36)	(0, 1)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
	seg		(40, 44, 48)	(0.72, 0.8, 0.88)	(0, 1)	(512, 536, 560)	(4, 5)	(0.5, 0.75)

Table 1. The search space of Poly-PC (base). For simplicity, we put the search space of stem channel into stage 1.

		Stem Channel	Neigh Num	Group Radius	Res-MLP Num	Output Channel	Expansion rate	Reduction rate
stage1	cls	(64, 72, 80)	(40, 48, 56)	(0.09, 0.1, 0.11)	(1, 2)	(128, 136, 144)	(4, 5)	(0.5, 0.75)
	det	(128, 136, 144)	(56, 64, 72)	(0.18, 0.2, 0.22)	(1, 2)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
	seg	(32, 40, 48)	(48, 56, 64)	(0.09, 0.1, 0.11)	(1, 2)	(64, 72, 80)	(4, 5)	(0.5, 0.75)
stage2	cls	-	(40, 48, 56)	(0.18, 0.2, 0.22)	(1, 2)	(128, 136, 144)	(4, 5)	(0.5, 0.75)
	det		(28, 32, 36)	(0.36, 0.4, 0.44)	(1, 2)	(512, 528, 544)	(4, 5)	(0.5, 0.75)
	seg		(48, 56, 64)	(0.18, 0.2, 0.22)	(1, 2)	(128, 136, 144)	(4, 5)	(0.5, 0.75)
stage3	cls	-	(40, 48, 56)	(0.36 ,0.4, 0.44)	(1, 2)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
	det		(12, 16, 20)	(0.72, 0.8, 0.88)	(1, 2)	(512, 528, 544)	(4, 5)	(0.5, 0.75)
	seg		(48, 56, 64)	(0.36, 0.4, 0.44)	(2, 3)	(256, 272, 288)	(4, 5)	(0.5, 0.75)
stage4	cls	-	(40, 48, 56)	(0.36, 0.4, 0.44)	(1, 2)	(512, 536, 560)	(4, 5)	(0.5, 0.75)
	det		(6, 8, 10)	(1.04, 1.2, 1.36)	(1, 2)	(512, 528, 544)	(4, 5)	(0.5, 0.75)
	seg		(48, 56, 64)	(0.72, 0.8, 0.88)	(2, 3)	(512, 536, 560)	(4, 5)	(0.5, 0.75)

Table 2. The search space of OFAT-PC (large). For simplicity, we put the search space of stem channel into stage 1.

Method	Points	Flops (G)	Params Count	OA
PointNet++ [10]	1k	3.2	100%	77.9
PointCNN [6]	1k	-	100%	78.5
DGCNN [16]	1k	2.4	100%	78.1
DRNet [12]	1k	-	100%	80.3
GBNet [13]	1k	-	100%	80.5
PRANet [2]	1k	-	100%	82.1
MVTN [4]	1k	1.8	100%	82.8
PointMLP [7]	1k	-	100%	85.4
Poly-PC (base)	**1k**	**3.3**	**88%**	**86.8**
Poly-PC (large)	**1k**	**5.8**	**50%**	**87.9**

Table 3. Shape classification results on the ScanObjectNN dataset.

by using the gradient on each GPU. Note that when optimizing a task with multiple GPUs, the gradients of the task on these GPUs would be averaged and then the task-specific parameters are updated accordingly.

C.2. Evolution search.

After obtaining the trained Poly-PC, we perform an evolution search to obtain the optimal subnets for each task. Subnets are evaluated and picked according to the manager of the evolution algorithm. Our objective here is to maximize the proxy score performance for each task while minimizing the model size (parameters). The proxy score for 3D shape classification is overall accuracy, for 3D semantic segmentation is mean classwise intersection over union (mIoU), and for 3D object detection is mean average precision (mAP) at 0.25 threshold, i.e., mAP@0.25. At the beginning of the evolution search, we pick 50 random architecture as seeds. The top 10 architectures are picked as parents to generate the next generation by crossover and mutation. For a crossover, two randomly selected candidates are picked and crossed to produce a new one during each generation. For mutation, a candidate mutates its depth with probability P_d first. Then it mutates each block with a probability of P_m to produce a new architecture. We set P_d to 0.2 and P_m to 0.4 in all our experiments.

D. Incremental learning for ScanObjectNN

In this part, we illustrate that Poly-PC is designed to enable incremental learning. We add the real-world object classification dataset ScanObjectNN into Poly-PC. ScanObjectNN is comprised of approximately 15k real scanned objects categorized into 15 classes with 2,902 distinct object instances. Existing point cloud analysis methods are significantly hampered by occlusions and noise of ScanObjectNN. Following [11], we experiment on PB_T50_RS, the hardest and most commonly used variant of ScanObjectNN. When Poly-PC is trained on the three datasets in the main paper, we fit such new task with task-specific parameters while

freezing all task-shared parameters. The results are reported in Tab. 3. The training recipe is given as: AdamW optimizer with weight decay 0.05, initial learning rate 0.008 with cosine annealing, a batch size of 16, and total epochs 100. Poly-PC (base) and Poly-PC (large) surpass existing methods by non-trivial margins in terms of overall accuracy (OA) while using fewer model parameters. Specifically, Poly-PC (base) surpasses baseline PointNet++ [10] 8.9 units and PointMLP [7] 1.4 units in terms of OA, demonstrating the efficacy of Poly-PC. Moreover, since a substantial piece of the network (i.e., the backbone) is shared among tasks, we only needs extra 88% parameters for Poly-PC (base) and 50% parameters for Poly-PC (large), further demonstrating that Poly-PC scales up more gracefully as the number of tasks increases.

E. Details on Poly-PC Architecture

E.1. Network input

ModelNet40 and ScanObjectNN. For 3D point classification, Poly-PC(base/large) takes a randomly subsampled point cloud as input, with a size of $N_{cls} \times C^{in}_{cls}$, where N_{cls} is the number of sampled points and is set to 1024, and $C^{in}_{cls} = 6$ denotes the input coordinate and the normalized coordinate of each point.

S3DIS. For semantic segmentation, Poly-PC(base/large) takes a randomly subsampled point cloud as input, with a size of $N_{seg} \times C^{seg}_{in}$, where is the number of sampled points and is set to 4096 for Poly-PC (base) and 16384 for Poly-PC (large), and $C^{in}_{seg} = 9$ denotes the input coordinate, color and normalized input coordinate of each point.

SUNRGBD. For object detection, Poly-PC (base/large) takes a randomly sampled point cloud of a SUNRGB-D depth image with a size of $N_{det} \times C^{in}_{det}$, where N_{det} set to 20k is the number of sampled points, and $C^{in}_{det} = 4$ the input coordinate and its height (distance to floor). The floor height is estimated as the 1% percentile of heights of all the points.

E.2. Subsample points number in each Res-SA layer

Poly-PC comprises four Res-SA layers to extract point features, with each Res-SA layer employing the furthest points sampling method to downsample the number of input points. For Poly-PC (base) in point classification, the four Res-SA layers downsample the points number to 512, 256, 64 and 16, respectively. For Poly-PC (large) in point classification, the four Res-SA layers downsample the points number to 512, 256, 128 and 64, respectively. For Poly-PC (base) in point segmentation, the four Res-SA layers downsample the points number to 1024, 256, 64 and 16, respectively. For Poly-PC (large) in point segmentation, the four Res-SA layers downsample the points number to 1024×4, 256×4, 64×4 and 16×4 respectively. For both Poly-PC (base) and Poly-PC (large) in point object detection, the four Res-SA layers downsample the points number to 2048, 1024, 512 and 256, respectively.

Layer name	input layer	layer params
classification		
SA1	raw point cloud	(512, 0.1, [32,32,64])
SA2	SA1	(256, 0.2, [64,64,128])
SA3	SA2	(64, 0.4, [128,128,256])
SA4	SA3	(16, 0.4, [256,256,512])
segmentation		
SA1	raw point cloud	(1024, 0.1, [32,32,64])
SA2	SA1	(256, 0.2, [64,64,128])
SA3	SA2	(64, 0.4, [128,128,256])
SA4	SA3	(16, 1.8, [256,256,512])
detection		
SA1	raw point cloud	(2048, 0.2, [64,64,128])
SA2	SA1	(1024, 0.4, [128,128,256])
SA3	SA2	(512, 0.8, [128,128,256])
SA4	SA3	(256, 1.2, [128,128,256])

Table 4. Backbone network architecture: layer details for each task.

E.3. Structures for each task

Fig. 2 and Fig. 3 plot the architectures for all tasks searched by Poly-PC.

F. Implementation details in ablation study

The effect of different proposed modules. We start with the baseline that uses the naive SA layer in Poly-PC to train multiple tasks. In this section, we provide the following additional details about the baseline: The backbone network employs four set abstraction layers, with layer parameters shown in Tab. 4. Each set abstraction (SA) layer has a receptive field determined by a ball-region radius r, an MLP network for point feature transform $MLP[c_1, ..., c_k]$ where c_i is output channel number of the i-th layer in the MLP. The SA layer also subsamples the input point cloud with farthest point sampling to n points. Each SA layer is specified by $(n, r, [c_1, ..., c_k])$ as shown in Tab. 4. For detection, we use two feature propagation (FP) layers to upsample the point features by interpolating the features on input points to output points. For segmentation, we use four feature propagation (FP) layers. We position the feature propagation layer in the head of each task and treat them as task-specific parameters. Experimentally, we optimize these three tasks using the same training protocol as that introduced in the main work for training supernets. Throughout all ablation trials, we enable all tasks to share the same proportion of parameters as the main results.

G. Limitation

In this paper, we present Poly-PC, a unified framework that simultaneously learns numerous point cloud tasks un-

der distinct dataset domains. Poly-PC is storage-efficient since multiple models with the vast majority of shared parameters in the backbone can be deposited into a single one. However, from the experiments, we observe that the parameters of head network for some tasks are relatively large, yet Poly-PC does not consider the parameter sharing in the head for all tasks. In the future, we will build an encoder-decoder structure that enables all tasks to share parameters throughout the network rather than only backbone network.

References

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR, 2018.

[2] Silin Cheng, Xiwu Chen, Xinwei He, Zhe Liu, and Xiang Bai. Pra-net: Point relation-aware network for 3d point cloud analysis. *IEEE Transactions on Image Processing*, 30:4436–4448, 2021. 2

[3] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European conference on computer vision*, pages 544–560. Springer, 2020.

[4] Abdullah Hamdi, Silvio Giancola, and Bernard Ghanem. Mvtn: Multi-view transformation network for 3d shape recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1–11, 2021. 2

[5] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, pages 367–377. PMLR, 2020.

[6] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018. 2

[7] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. In *International Conference on Learning Representations*, 2021. 2, 3

[8] Houwen Peng, Hao Du, Hongyuan Yu, Qi Li, Jing Liao, and Jianlong Fu. Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. *Advances in Neural Information Processing Systems*, 33:17955–17964, 2020.

[9] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[10] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 2, 3

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

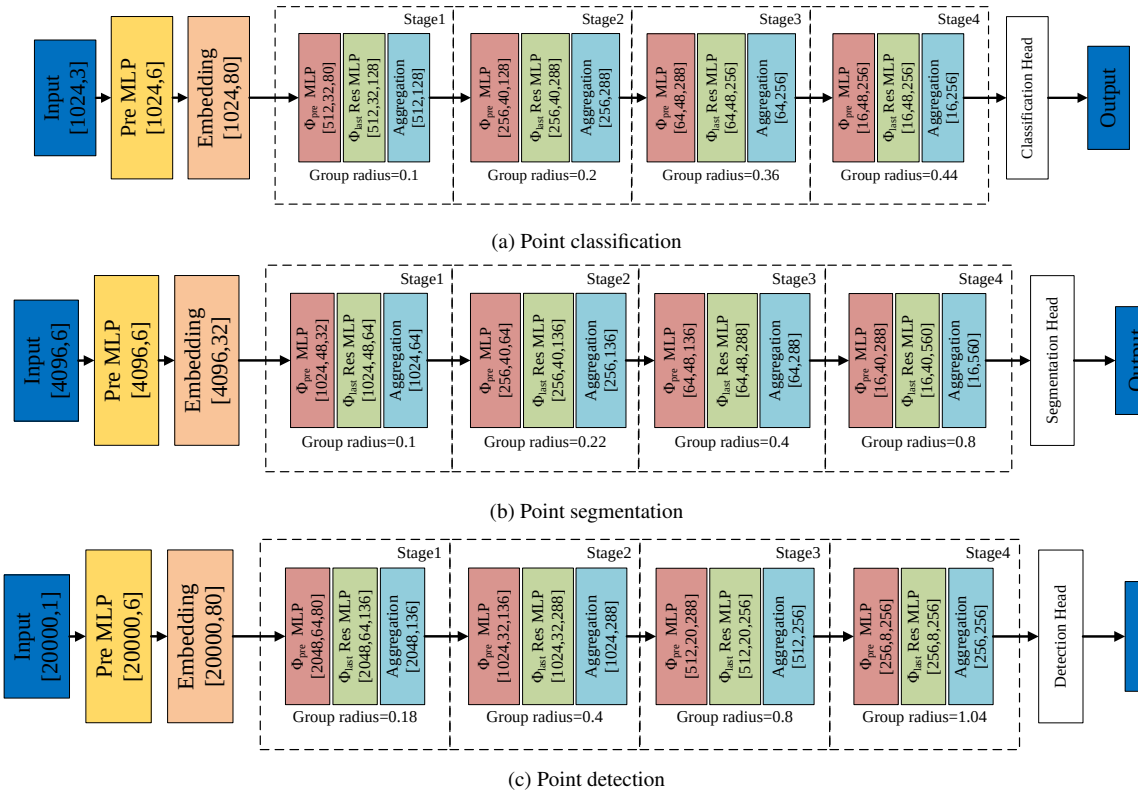
[12] Shi Qiu, Saeed Anwar, and Nick Barnes. Dense-resolution network for point cloud classification and segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3813–3822, 2021. 2

[13] Shi Qiu, Saeed Anwar, and Nick Barnes. Geometric back-projection network for point cloud classification. *IEEE Transactions on Multimedia*, 24:1943–1955, 2022. 2

[14] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[15] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[16] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 2

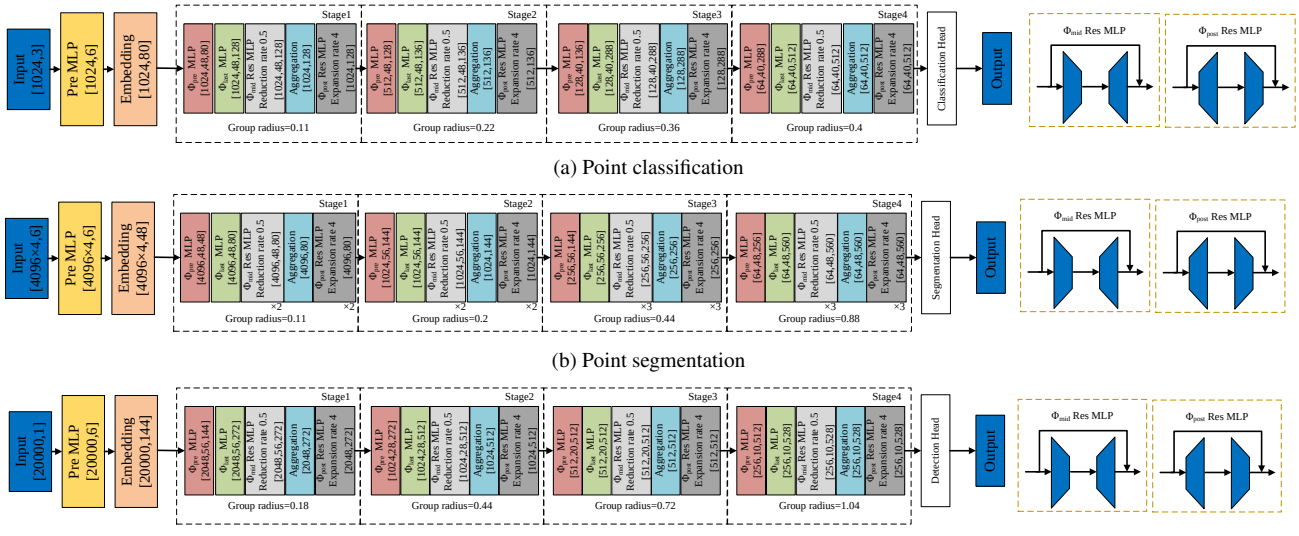


(a) Point classification

(b) Point segmentation

(c) Point detection

Figure 2. Architectures of Poly-PC (base) for point classification, segmentation and detection. All tasks share weights of their common parts in each layer of the backbone.



(a) Point classification

(b) Point segmentation

(c) Point detection

Figure 3. Architectures of Poly-PC (large) for point classification, segmentation and detection. All tasks share weights of their common parts in each layer of the backbone.