PointNetv3: a Faster and More Accurate PointNet++

Anonymous Author(s)
Affiliation
Address
email

Abstract

Access to 3D point cloud representations has been widely facilitated by LiDAR sensors embedded in various mobile devices. This has led to an emerging need for fast and accurate point cloud processing techniques. In this paper, we revisit and dive deeper into PointNet++, one of the most influential yet under-explored networks, and develop faster and more accurate variants of the model. We first present a novel Separable Set Abstraction (SA) module that disentangles the vanilla SA module used in PointNet++ into two separate learning stages: (1) learning channel correlation and (2) learning spatial correlation. The Separable SA module requires far fewer FLOPs and is significantly faster than the vanilla version, yet it achieves comparable performance. We then introduce a new Anisotropic Reduction function into our Separable SA module and propose an Anisotropic Separable SA (ASSA) module that substantially increases the network's accuracy. We later replace the vanilla SA modules in PointNet++ with the proposed ASSA modules, and denote the modified network as PointNetv3. Extensive experiments on point cloud classification, semantic segmentation, and part segmentation show that PointNetv3 outperforms PointNet++ and other methods, achieving much higher accuracy and faster speeds. In particular, PointNetv3 outperforms PointNet++ by 7.0 mIoU on S3DIS Area 5, while maintaining $1.6\times$ faster inference speed on a single NVIDIA 2080Ti GPU. Our scaled PointNetv3 (L) variant achieves 66.8 mIoU and outperforms KPConv, while being more than $54\times$ faster.

1 Introduction

Among the various 3D object representations, point clouds have been surging in popularity, becoming one of the most fundamental 3D representations. This popularity stems from the increased availability of 3D sensors, like LiDAR, which produce point clouds as their raw output. The growing presence of point cloud data has been accompanied by the development of many 3D deep learning methods [30, 42, 21, 39, 24]. Even though these methods achieve impressive performance, they are generally expensive in terms of computational cost (Figure 1). With the integration of LiDAR sensors into hardware constrained devices, such as mobile devices and AR headsets, there is a growing interest in efficient models for point cloud processing. Given the limited computational power of mobile devices and embedded systems, the design of mobile-friendly point cloud-based algorithms should not only focus on providing good accuracy, but also on maintaining high computational efficiency.

When processing point cloud data, one can always opt to convert the data into representations more accessible to deep learning frameworks. Popular options are multi-view methods [35, 5, 43] and voxel-based methods [6, 48]. Converting to these representations generally requires additional computations and memory, and can lead to geometric information loss [25]. It is therefore more desirable to operate directly on point clouds. To that extent, we are currently witnessing a surge in new point-based methods [29, 30, 42, 21, 39, 24]. The first of such methods was introduced by Qi *et al*. through the seminal PointNet [29] architecture. PointNet operates directly on point clouds,

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

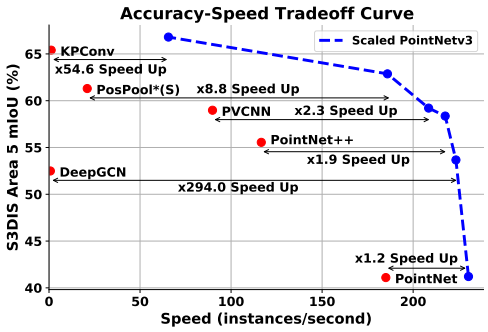


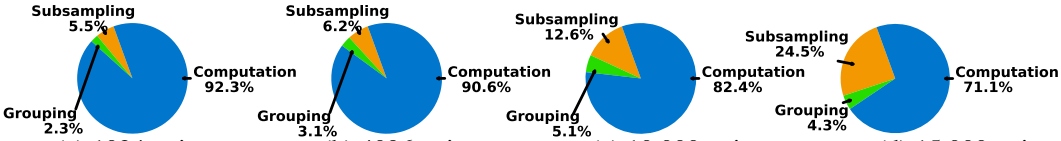
Figure 1: **Tradeoffs between accuracy (mIoU on S3DIS Area-5) and inference speed (instances/second)**. Speed is reported as the mean value of 200 runs on a single GTX 2080Ti GPU. PointNetv3 scaled with different widths and depths shown in -·- outperforms the state-of-the-art methods in ● with better accuracies and faster speeds. Refer to Section 5.2 for details.

without the need for an intermediate representation. Despite its efficiency, PointNet merely learns per-point features individually and discards local information, which restrains its performance. As a variant of PointNet, PointNet++ [30] presents a novel Set Abstraction module that sub-samples the point cloud, groups the neighborhood, extracts local information via a set of multi-layer perceptrons (MLPs), and then aggregates the local information by a reduction layer (*i.e*. pooling). Figure 1 shows how PointNet++ outperforms the pioneering PointNet [29] by a large margin. PointNet++ also obtains better accuracy than the graph-based method DeepGCN [21], and does so with a $100\times$ speed gain. PointNet++ provided a good balance between accuracy and efficiency, and was therefore widely utilized in various tasks like normal estimation [8], segmentation [28, 16], and object detection [33]. After PointNet++, graph-based [34, 40, 42, 21], pseudo-grid based [38, 22, 27, 39] and adaptive weight-based [41, 23, 7, 45], surged as state-of-the-art in point cloud tasks. Nearly all these methods improve performance at the cost of speed, as shown in Figure 1. In this work, we focus on designing point cloud networks that are both fast and accurate. Inspired by its success, both in terms of the accuracy-speed balance and its wide adoption, we take a deep dive into PointNet++. We conduct an extensive analysis of its architectural design (refer to Section 3.1) and latency decomposition (Figure 2). Interestingly, we demonstrate that both its efficiency and accuracy can be improved sharply by minimal modifications to the architecture. These modifications lead to a new architecture design that is faster and more accurate than currently available point methods (shown in -·- in Figure 1).

Contributions. (1) We demonstrate that the MLPs performed on the neighborhood features in the Set Abstraction (SA) module of PointNet++ hamper the inference speed. We greatly accelerate this module by introducing a new **separable SA** module that processes on point features directly. **(2)** We discover that all operations for processing neighbors in the SA module are isotropic and limit its accuracy. We propose a novel **Anisotropic Reduction** layer that treats each neighbor differently. We then insert Anisotropic Reduction into our Separable SA and propose the Anisotropic Separable Set Abstraction (ASSA) module that significantly increases accuracy. **(3)** We present PointNetv3, in which we replace the vanilla SA with our ASSA. PointNetv3 shows a much higher accuracy and faster speed compared to PointNet++ and previous methods on various tasks (point cloud classification, semantic segmentation, and part segmentation). We further study two regimes for up-scaling PointNetv3. As shown in Figure 1, *our scaled PointNetv3 outperforms the previous state-of-the-art with a much faster inference speed*. In particular, scaled PointNetv3 achieves better accuracy than the graph-based method DeepGCN [21] with an increase in speed of $294\times$, the pseudo grid-based method KPConv [39] ($54\times$ faster), the adaptive weight-based method PosPool*(S) [24] ($9\times$ faster), and the efficient 3D method PVCNN [25] ($2.3\times$ faster). Code is available in **Appendix**.

2 Related Work

Projection-based methods. Due to the unstructured nature of point clouds, convolutional neural networks (CNNs) that tend to work impressively well on grid stuctured data (*e.g*. images, texts and videos) fail to apply directly on point clouds. One common solution for processing point clouds is to project them into collections of images (views) [35, 5, 43] or 3D voxels [6, 48, 37]. Common CNN backbones (using 2D or 3D convolutions) can be subsequently utilized to perform these intermediate representations. Although projection-based methods allow for utilizing the well studied convolutional neural networks to point-cloud applications, they are computationally expensive as they are associated with the additional cost of constructing intermediate representations. Moreover, the projection of point clouds causes loss of important geometric information [25].



(a) 1024 points (b) 4096 points (c) 10,000 points (d) 15,000 points

Figure 2: **Latency Decomposition of PointNet++.** We show the inference run time decomposition of PointNet++ under different numbers of points as input on one NVIDIA GTX2080Ti GPU. The actual computation consumes over 70% in all the cases, showing that the actual computation part is the major speed bottleneck.

Point-based methods. Pioneering work explored the possibility of processing point clouds directly. Qi *et al*. proposed PointNet [29] that leverages point-wise MLPs to extract per point features individually. To better encode locality, Qi *et al*. further presented Set Abstraction (SA) to aggregate features from the points' neighborhood, and a hierarchical architecture named PointNet++ [30] that learns multilevel representations and reduces computations. After PointNet++, numerous point-based methods considering neighborhood information were proposed. Graph-based methods [34, 17, 42, 40, 21, 20] represent point clouds as graphs and process point clouds with graph neural networks. Pseudo grid-based methods project neighborhood features onto different forms of pseudo grids such as tangent planes [38], grid cells [12, 47, 22, 27, 39] and spherical grid points [50] which allow convolving with regular kernel weights like CNNs. Adaptive weight-based methods perform weighted neighborhood aggregation by considering the relative positions of the points [41, 23, 7, 24] or point density [45]. These methods rely either on designing sophisticated and customized modules which usually require expensive parameter tuning for different applications [39, 22, 26], or on performing expensive graph kernels [42, 21] that achieve better performance than PointNet and PointNet++ at the expense of computational complexity.

Efficient Neural Networks. Efficient neural networks is a class of architectures that target mobile and embedded systems applications. These networks are usually designed to provide a balance between accuracy and efficiency (*e.g*. latency, FLOPs, memory, and power). MobileNet [10] utilizes depth-wise separable convolutions to reduce the required FLOPs and latency of a regular CNN for image processing. Depth-wise separable convolutions disentangle convolutions into learning channel correlations using point-wise convolutions and learning spatial correlations using depth-wise convolutions. Other efficient neural networks usually leverage either depth-wise separable convolutions with better designed architectures to improve performance [32, 4, 49] or study new efficient operations to replace the regular convolutions [26, 44]. In 3D, efficient neural networks include ShellNet [50], PVCNN [25], Grid-GCN [46], RandLA-Net [11], SegGCN [19] and LPNs [18]. ShellNet [50] and SegGCN [19] speed up the pseudo grid-based methods by aggregating neighborhood features through efficient 1D convolutions or fuzzy spherical convolutions on the predefined pseudo grids like shells. PVCNN [25] and Grid-GCN [46] reduce the time spent in querying neighborhood by combining voxelization in point based methods. RandLA-Net [11] reduces the subsampling complexity by leveraging random sampling and further improves the speed by operating on a large-scale point cloud directly without chunking. LPN [18] improves the speed of convolving neighborhood features by a simple group wise matrix multiplication. Nevertheless, all the efficient methods mentioned above require performing convolutions on neighborhood features, which we deem through extensive experiments as unnecessary. Therefore, our algorithm achieves much faster speeds compared to these methods (ref to Section 4). It is also worthwhile to mention that our method can be made even faster with the voxelization trick in PVCNN and Grid-GCN to further reduce the latency of neighborhood querying. We leave that as future work and focus on our own contributions.

3 Methodology

3.1 Preliminary: PointNet++

PointNet++ [30] improves PointNet [29] by providing two main contributions: (1) Developing a U-Net [31] like architecture to process a set of points, which are sampled in a metric space in a hierarchical fashion. This mechanism captures multi-scale features and reduces the required

computation. (2) Developing a Set Abstraction (SA) module to process and abstract the locality from the local neighbors to a new set of points with fewer elements. The SA module is used as the basic building block to be stacked to form the backbone of PointNet++.

Analysis of the Set Abstraction Module. The vanilla SA module proposed in PointNet++ consists of two parts: point subsampling and feature aggregation, as illustrated in Figure 3(a). The subsampling layer takes a point cloud $X = \{P, F\}$ as an input and leverages iterative farthest-point sampling to acquire X', a subset of X. P and F denote the coordinates and features, respectively. The feature aggregation block is built for learning locality from local neighbors and is composed of a grouping layer, an MLP block, and a reduction layer. The grouping layer obtains the neighborhood composed of K neighbors for each point in X' using the ball query, with X as the support set. The resulting point neighborhood is denoted as $\mathcal{N}(X')$. The MLP block consists of L layers of MLPs, and each MLP is followed by a Batch Normalization (BN [13]) layer and a ReLU activation. By default, PointNet++ sets $L = 3$. The number of feature aggregation blocks inside one SA module, referred to as depth D in this paper, is set to $D = 2$. The reduction layer (*a.k.a*, pooling) aggregates the neighborhood information by a reduction function, *e.g*. mean, max, or sum. The feature aggregation is formulated as shown in Equation (1):

$$\mathbf{f}_i^{l+1} = \mathcal{R}\left(\left\{\mathrm{MLPs}((\mathbf{p}_j - \mathbf{p}_i)||\mathbf{f}_j^l)|j \in \mathcal{N}(i)\right\}\right), \tag{1}$$

where $\mathcal{R}$ is the reduction function across the neighborhood dimension, which is used for aggregating the neighborhood information. $\mathbf{p}_i$, $\mathbf{f}_i^l$, $\mathcal{N}(i)$ and $||$ denote the coordinates, the features in the l^{th} layer of the network, the neighborhood of the i^{th} point, and the concatenation operator across the channel dimension, respectively. The main issues with the vanilla SA module are: **(1)** the computational cost is unnecessarily high. *MLPs are unnecessarily performed on the neighborhood features*, which causes a considerable amount of latency in PointNet++. One straightforward remedy is to use MLPs to learn a feature embedding on the point features directly instead of doing so on the neighborhood features. This reduces the FLOPs of each MLP by a factor of K. **(2)** *All operations on neighbors are unnecessarily isotropic.* In other words, the MLPs and the reduction layer treat all local neighbors equally. This severely limits the representation capability of the network.

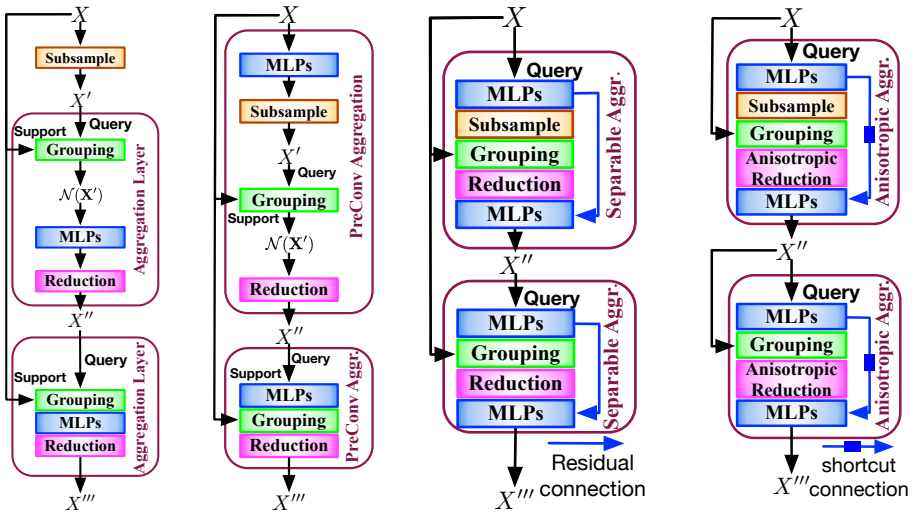
Latency Decomposition. Figure 2 shows the latency decomposition of PointNet++ [30] with different numbers of points as input. Here, the latency, which is the overall run time for the inference stage, was measured using a single Nvidia GeForce RTX 2080Ti GPU and one Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00GHz. We note here that latency is measured on the same hardware setting throughout this work. The latency of PointNet++ can be decomposed into three main contributing factors: (1) point subsampling, (2) grouping, (3) actual computations. The actual computations of PointNet++ mainly come from processing neighborhood features by MLPs shown in Equation 1. Note that we consider the time spent on data access implicitly in each part. Point clouds with four different input sizes were studied: 1024, 4096, $10,000$, and $15,000$. The first two input sizes are commonly encountered in classification tasks [2], and the last two are usually input sizes for patch-based segmentation [30, 39] and LiDAR-based object detection [33]. Clearly, computations contribute to the majority of latency (over 70 %), especially for small input sizes. This suggests that *the computational complexity could be the major speed bottleneck for networks involving PointNet++.*

3.2 Anisotropic Separable Set Abstraction (ASSA)

In this section, we gradually introduce the modified vanilla SA modules. Initially, we focus on speeding up vanilla SA. This is achieved through proposing two modules, namely, PreConv SA module and Separable SA module. Later, we focus our attention on improving the accuracy by proposing an Anisotropic SA module.

PreConv Set Abstraction module. We propose PreConv SA module to perform all MLPs before the grouping layer, as shown in Figure 3(b). PreConv SA performs the MLPs on the point features directly, and not on the K local neighbors. This reduces the required FLOPs by K times. PreConv SA speeds up PointNet++ by $\sim 55\%$ (15,000 points), as shown in Section 5.1. Additionally, PreConv SA is equivalent to vanilla SA in the case where the $(\mathbf{p}_j - \mathbf{p}_i)$ term is not included in Equation (1). We show the proof of this equivalence in the **Appendix**.

Separable Set Abstraction module. Next, we present Separable Set Abstraction, shown in Figure 3(c), which is more accurate than PreConv SA, yet requires the same latency. The main idea of



(a) Vanilla SA (b) PreConv SA (c) Separable SA (d) ASSA

Figure 3: **Comparison of proposed variants of the Set Abstraction (SA) module and the Vanilla SA module.** (a) Vanilla SA [30] applies MLPs on neighbor features. (b) The proposed PreConv SA applies MLPs on the point features directly. (c) Our Separable SA separates the MLPs to also process on the aggregated features from a local neighbors. (d) Our final ASSA module replaces the reduction layer in Separable SA with a new Anisotropic Reduction layer. X, X', X'', X''' are the input points, the subsampled points, the output of the first feature aggregation block, and the final output of the module, respectively. The shortcut layer is the residual connection with a linear mapping.

Separable SA is borrowed from depth-wise separable convolutions [10], where the regular convolution is split into one point-wise convolution (MLPs), one depth-wise convolution (channel shared convolution), and then another point-wise convolution. Separable SA evenly separates the MLPs before and after the reduction layer and further adds a residual connection between the outputs of the two parts of MLPs. *The main reasons why the Separable Set Abstraction module is better than PreConv are:* ***(1)*** *after reduction MLPs further process the aggregated neighborhood information*, which were previously missed in PreConv; ***(2)*** *The residual connection not only stabilizes training, but also provides better feature embedding.* We will later show why introducing residual connections leads to better embedding by relating such connections with recent findings in graph convolution field. The Separable SA module without the residual connection is similar to GCNConv proposed by Kipf [15], which merely learns the pooled (*e.g.* mean, max, sum) neighborhood information. The Separable SA module equipped with the residual connection is similar to SAGEConv proposed by Hamilton [9]. SAGEConv improves GCNConv by aggregating the neighborhood information and then adding it to the transformed node feature to learn a better embedding. Details of GCNConv and SAGEConv are available in the **Appendix**. Another minor change from PreConv SA to Spearable SA is that we query the neighborhood using the subsampled point cloud X' as the support set to further reduce computational complexity from the second aggregation block.

Anisotropic Separable Set Abstraction module. PreConv SA and Separable SA cut down computational complexity at the expense of accuracy, *e.g.* Separable SA leads to a reduction of 3 mIoU on S3DIS Area 5 compared to PointNet++ (Section 5.1). There are two reasons for the drop in accuracy. *First, geometric information is not well encoded in the current variants of the SA module.* The geometric information can be represented by any relative information (edge information) between the neighbor and the center, *e.g.* the relative position $(\mathbf{p}_j - \mathbf{p}_i)$ in PointNet++. The experiments in Section 5.1 show that geometric information is essential for point feature embedding. While vanilla SA module Equation (1) encodes geometric information fairly well by a set of MLPs, our two SA modules discard geometric information for the sake of efficiency. *Second, the reduction layer is an isotropic operation that treats each neighbor the same and thus leads to a sub-optimal representation.* Recall in a depth-wise separable convolution, the depth-wise convolution uses different weights to summarize features from the 3×3 receptive field. However, simply introducing the depth-wise

convolution kernel to point neighborhood aggregation does not work, as: (1) the neighbors are not necessarily ordered for the sake of efficiency; (2) the convolution kernel is shared by all the points and neighbors and leads to poor neighborhood aggregation where the local geometric varies. We propose an efficient geometric-aware *Anisotropic Reduction* layer to effectively aggregate the point neighborhood information. The term "Anisotropic" indicates that our reduction layer considers each neighbor differently. We insert Anisotropic Reduction into the separable SA module and present our final variant of the SA module, the Anisotropic Separable Set Abstraction (ASSA) module. ASSA is formulated as follows:

$$\begin{aligned} \mathbf{f}_i^{res} &= \mathrm{MLPs}(\mathbf{f}_i^l) \\ \mathbf{f}_i^{l+1} &= \mathbf{f}_i^{res} + \mathrm{MLPs}\left(\left\{\mathcal{R}\left(\frac{\Delta x_{ij}\mathbf{f}_j^{res}||\Delta y_{ij}\mathbf{f}_j^{res}||\Delta z_{ij}\mathbf{f}_j^{res}}{r}\right) | j \in \mathcal{N}(i)\right\}\right) \end{aligned} \tag{2}$$

$\Delta x_{ij} = x_j - x_i$, Δy_{ij} and Δz_{ij} are the relative positions between the neighbor j and the center i in the x, y, z dimension, respectively. The relative positions are used as scaling weights for aggregating the features across the neighborhood dimension, and they are normalized by the radius of the ball query r. The neighborhood features are scaled by the three corresponding relative positions individually. The three scaled neighborhood features are then concatenated together and passed into the reduction layer. To reduce the computational complexity caused by the concatenation of the three scaled features, we set the last MLP before reduction as a bottleneck layer. This layer reduces the number of channels by a factor of 3. The output of the reduction layer is then processed by another MLP block and is added to the output before reduction. Due to the channel mismatch, the output of before reduction MLPs is mapped by a linear layer (*a.k.a* the shortcut layer) before the addition. We highlight that our Anisotropic Reduction does not rely on any heuristic grouping (as done in PosPool [24]), and we make full use of the information from the neighborhood features. The pseudo code for ASSA in PyTorch-like style is available in the **Appendix**.

It is worth noting that all MLPs in ASSA are processed on the point features directly, not on the neighborhood, which greatly reduces the computations compared to Equation (1). In particular, for one aggregation block with $L = 3$ MLPs, ASSA reduces the FLOPs consumed in vanilla SA by: $\frac{C\times C\times N\times K\times L}{C\times C\times N\times L+3\times C\times N\times K+3\times C\times N} \approx K$ times. Typically, K is around 32. All of our SA variants are permutation invariant, which favors 3D deep learning on point clouds. More details of the ASSA module and its comparison with previous modules are provided in the **Appendix**.

3.3 PointNetv3

We replace the vanilla SA module in PointNet++ [30] with our proposed ASSA module. The other parts are kept the same as PointNet++, including the number of SA modules (4), the number of aggregation blocks in SA ($D = 2$), the layers of MLPs in an aggregation block ($L = 3$), the channel sizes, the neighborhood querying configurations (ball query algorithm with maximum neighborhood size K and radius r) and the subsampling configurations (farthest point sampling). The modified architecture of PointNet++ is referred to as PointNetv3, the third version of PointNet [29]. Section 4 will show that PointNetv3 can achieve much higher accuracies compared to PointNet and PointNet++ and is indeed faster on various vision tasks.

3.4 Scaling PointNetv3

Since the PointNetv3 is much faster than both PointNet++ [30] and the state-the-of-art networks, we now present two ways to up-scale PointNetv3 to improve its accuracy. We consider two scaling regimes: width scaling and depth scaling. We show the performance of each scaling regime in the ablation study presented in Section 5.2.

Width Scaling Regime. In width scaling regime, we modified the channel size of PointNetv3. PointNetv3 is built upon PointNet++ [30], which uses hand-crafted channel size for each convolution layer. To make the scaling more programmable and user-friendly for scaled PointNetv3, we make the output of each feature aggregation block inside one ASSA module to have the same channel size, and concatenate them as the output of the module. After this modification, we can easily study the effect of width scaling on the accuracy and the speed, by simply changing the initial channel size C. **Depth Scaling Regime.** The second way to scale the architecture is to increase the depth of the network. Since a single isotropic aggregation block in the ASSA module is analogous to a single layer of

depthwise separable convolution (Section 3.2), we controls the depth scaling by changing the number of aggregation blocks D stacked in the ASSA module. D is set to 2 by default in PointNetv3. We can decrease D to 1 to make PointNetv3 faster or increase D to improve its accuracy. Among all width or depth scaled versions of PointNetv3, we emphasize **PointNetv3 (L)**, a large PointNetv3 network with $C = 128$ and $D = 3$. In most of the experiments, we compare PointNetv3 and PointNetv3 (L) with the state-of-the-art.

4 Experiments

We studied the accuracy and speed of PointNetv3 and PointNetv3 (L) on S3DIS semantic segmentation [1], ShapeNet part segmentation [3], and ModelNet40 point cloud classification [2]. To enable a fair comparison, the same data processing and evaluation protocols adopted by the state-of-the-art method PosPool [24] were used in our experiments.

Methods	**mIOU** %	**Inference Speed** instances/second
PointNet [29]	41.1	185.0
DeepGCN [21]	52.5	0.8
PointCNN [22]	57.3	124.1
Grid-GCN [46]	57.8	123.5
PVCNN [25]	59.0	89.8
PosPool*(S) [24]	61.3	21.0
SegGCN [19]	63.6	29.3
KPConv [39]	65.4	1.2 (24.2)
PosPool* [24]	66.7	8.3
PointNet++ [30]	55.6	116.6
PointNetv3	62.6 (+7.0)	188.6 (1.6×)
PointNetv3 (L)	66.8 (+11.3)	65.6

Table 1: **S3DIS scores (mIoU) on Area-5.** PointNetv3 outperforms PointNet++ and other methods with much higher accuracy and faster speed. PointNetv3 (L) performs better than the state-of-the-art KPConv [39] and PosPool* [24] while being over 7.9× faster.

4.1 3D Scene Segmentation

Setups. We conducted extensive experiments on the Stanford large-scale 3D Indoor Spaces (S3DIS) dataset [1]. Following [22, 25, 24], we trained all our models on Area 1, 2, 3, 4, and 6 and tested them on Area 5. We optimized all of our networks using SGD with weight decay 0.001, momentum 0.98 and initial learning rate (LR) 0.02. We trained 600 epochs and used an exponential LR decay. At each inference time, a single RTX 2080Ti GPU was used to measure the speed for each method using a batch size of 16; each item in the batch has $15,000$ points ($16 \times 15,000$). If the batch size was too large to feed into the GPU, we lowered the batch size. *Note that we focus on the speed since FLOPs and the model parameter size are not indicative of the actual latency [26, 25].* The inference speed is calculated as the number of instances evaluated in one second (ins./sec.). The average speed over 200 runs is reported. Other methods were measured in similar manner. Note KPCOnv [39] has to compute the pseudo kernels for each point cloud during data preprocessing. For a fair comparison, we show the speed of calculating the pseudo kernels on the fly. We also include the speed of KPConv with preprocessed pseudo kernels in $()$ in the table.

Comparison with state-of-the-art. Table 1 compares the proposed PointNetv3 and PointNetv3 (L) with PointNet++ [30] and the state-of-the-art on S3DIS. *PointNetv3 outperforms PointNet++ by* 7 *mIoU and is* $1.6\times$ *faster.* PointNetv3 also achieves much better accuracy than the two efficient point cloud processing algorithms PVCNN [25] and Grid-GCN [46], while also being over $1.5\times$ faster. *PointNetv3 (L) achieves state-of-the-art performance with a mIoU of* 66.8% *on S3DIS, with very high speed.* PointNetv3 (L) is $294\times$ faster than the graph-based method DeepGCN [21], $54.6\times$ faster than the state-of-the-art pesudo grid-based method KPConv [39], $7.9\times$ faster than the state-of-the-art adaptive weight-based method PosPool* [24], and $2.2\times$ faster than the best-performing efficient method SegGCN [19]. Note that PosPool* refers to PosPool with sinusoidal position weight, and that PosPool* (S) denotes the small model.

4.2 3D Object Classification

Setup. As a common practice, we benchmark PointNetv3 on the ModelNet40 [2] object classification dataset. We adopted a similar training setting as that on S3DIS except that we used LR 0.001 and a cosine LR decay in this experiment. At the inference time, a single RTX 2080Ti GPU was used to measure the speed for the classification task using $16 \times 10,000$ points as input.

Methods	OA %	Inference Speed instances/second
PointNet [29]	89.2	483.8
SpiderCNN [47]	90.5	< 275.7
PointCNN [22]	92.5	183.4
PosPool*(S) [24]	92.6	48.8
DGCNN [42]	92.9	11.6
KPConv [39]	92.9	(30.1)
Grid-GCN [46]	93.1	172.0
PosPool* [24]	93.2	27.6
PointNet++ [30]	90.7	275.7
PointNetv3	92.7 (+2.0)	586.4 (2.1×)
PointNetv3 (L)	93.0 (+2.3)	153.2

Table 2: **Comparison of our PointNetv3 and PointNetv3 (L) with other methods on ModelNet40 point cloud classification.** PointNetv3 outperforms PointNet++ with 2.0 higher overall accuracy (OA) than PointNet++ and is 2.13 times faster . PointNetv3 (L) achieves on par accuracy with the state-of-the-art while maintaining a high speed.

Methods	mIoU %	Inference Speed instances/second
PointNet [29]	83.7	1883.5
PosPool* (S) [24]	85.1	107.7
DGCNN [42]	85.2	151.4
LPN [18]	85.7	190.6
PosPool* [24]	85.8	58.0
PointCNN [22]	86.1	626.4
RS-CNN [23]	86.2	<350.4
KPConv [39]	86.2	(56.3)
PointNet++ [30]	85.1	350.4
PointNetv3	85.4 (+0.3)	782.5 (2.2×)
PointNetv3 (L)	86.1 (+1.0)	438.5 (1.3×)

Table 3: **Comparison of the part-averaged IoU (mIoU) of our PointNetv3 and PointNetv3 (L) with other methods on ShapeNetPart part segmentation.** Both of PointNetv3 and PointNetv3 (L) outperform PointNet++ with a higher speed. PointNetv3 (L) achieves a comparable accuracy as the state-of-the-art while being much faster.

Comparison with state-of-the-art. Table 2 compares PointNetv3 and PointNetv3 (L) with the state-of-the-art. PointNetv3 outperforms PointNet++ by 2 units in overall accuracy and is $2.1\times$ faster than PointNet++. PointNetv3 (L) achieves on par accuracy as the state-of-the-art methods KPConv [39] and PosPool* [24] while being $5.0\times$ and $4.4\times$ faster, respectively.

4.3 3D Part Segmentation

Data. ShapeNetPart is a commonly used benchmark for 3D part segmentation. The networks were optimized using Adam [14] with momentum 0.9. The other training parameters were the same as ModelNet40 experiment. The speed of each method was measured with an input of 16×2048 points. We report the part-averaged IoU (mIoU) as the evaluation metric for accuracy.

Comparison with state-of-the-art. Table 3 shows that PointNetv3 again outperforms PointNet++ with a sharp increase ($2.2\times$) in speed on the ShapeNetPart part segmentation dataset. PointNetv3 (L) also achieves 1 unit higher mIoU than PoinetNet++ with a $1.3\times$ faster speed. Additionally, PointNetv3 (L) attains on par accuracy, 86.1% mIoU, with the state-of-the-art and is much faster. For example, PointNetv3 (L) is nearly $7.8\times$ faster than KPConv [39].

5 Ablation Study

An ablation study was conducted on S3DIS [1] Area-5. We show the effectiveness of the proposed SA variants and the effect of the two scaling regimes on the performance of PointNetv3.

5.1 Ablation on Proposed SA variants

Aggregation	mIoU %	Speed ins./sec.
Vanilla SA	55.6	116.6
PreConv SA	48.7	180.9
Separable SA (SSA)	52.4	180.0
SSA + Attentive Pooling[11]	59.0	142.0
SSA + PosPool[24]	62.0	168.4
Anisotropic Separable SA	**62.6**	**188.6**

Table 4: **Ablation study of the proposed SA variants.** All proposed SA variants achieve a faster speed than the vanilla SA. Our ASSA further improves the accuracy of the Separable SA module and outperforms other methods, while also being faster.

The proposed PreConv SA and Separable SA modules achieve faster inference speeds. Table 4 shows the speed and the accuracy of the proposed PreConv Set Abstraction (SA) module, the Separable SA module, and the Anisotropic Separable SA (ASSA) module compared to the vanilla SA. All of our proposed SA modules lead to a sharp increase (over $1.6\times$) in inference speed. The proposed Separable SA module can boost the accuracy of PreConv by 3.7 mIoU, which verifies the effectiveness of separable MLPs and residual connections. Comparing the ASSA

module with the Separable SA module, one can clearly see the importance of encoding the geometric information and the effect of the anisotropic operation to achieve higher accuracy. Additionally, we provide a comparison of our proposed Anisotropic Reduction with the Attentive Pooling used in RandLA-Net [11] and the PosPool proposed in [24]. Our method clearly outperforms both of these methods in terms of accuracy and inference speed.

5.2 Ablation on Scaling Regimes

We now study the effects of ablating the width and depth of a network on its accuracy and inference speed. The initial channel size of the network is referred to as width (denoted by C), whereas the number of aggregation layers inside a single SA module is referred to as depth (denoted by D).

Width scaling. Figure 4 (left) shows the effect of the width scaling regime. When the width of the network is small, increasing the width leads to a significant improvement in accuracy. For example, simply increasing the width C from 3 to 8 sharply improves the accuracy from 41.21 mIoU to 53.95 with a negligible drop in inference speed. However, when the network is wide enough ($C \geq 128$), increasing the width further only leads to a marginal improvement in accuracy, yet reduces the inference speed noticeably.

Depth scaling. Figure 4 (right) shows the effect of the depth scaling regime. We study the depth scaling with $C = 128$, which is the sweet point of width scaling. When the network is shallow, with a depth of $D \leq 3$, increasing the depth leads to an obvious increase in accuracy, similarly to width scaling. However, depth scaling rapidly saturates as the depth increases. Additionally, depth scaling leads to a linear reduction in speed.

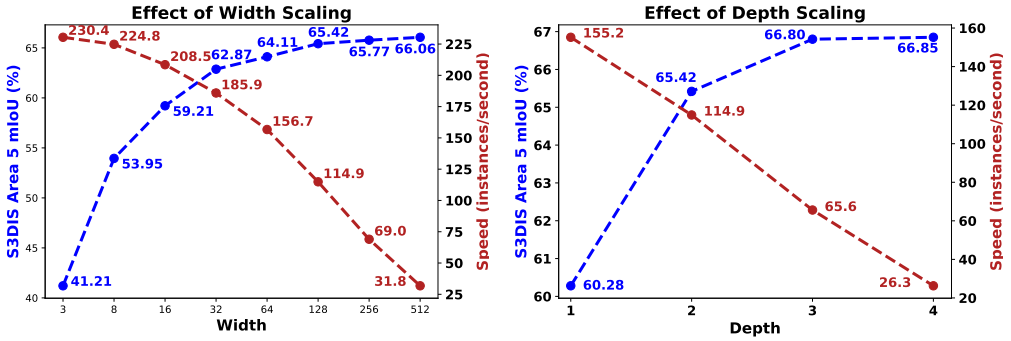


Figure 4: **Effect of Width (left) and Depth Scaling (right).** Increasing either the width (the initial channel size) or the depth (the number of aggregation layers in a single SA module) leads to an improvement in accuracy and drop in inference speed.

6 Conclusion

In this paper, we dove deeper into the architecture of PointNet++. We noticed that PointNet++ suffers from a computational burden attributed to the MLPs that process the neighborhood features in the set abstraction (SA) module. We also found out that the accuracy of PointNet++ is limited by the isotropic nature of its operations. To solve these issues, we proposed a PreConv SA module, a Separable SA module, and finally an Anisotropic Separable SA (ASSA) module that aim to reduce the computational cost and improve the accuracy. We then replaced the vanilla SA module in PointNet++ with our ASSA module, and proposed a fast and accurate architecture, namely, PointNetv3. Extensive experiments were conducted to verify the presented claims, and showed that PointNetv3 achieves largely improved accuracy and much faster speed on various point cloud tasks, such as classification, semantic segmentation, and part segmentation. We also studied up-scaling PointNetv3. The scaled PointNetv3 set new state-of-the-art on various tasks with faster speeds. For future work, one could leverage both random sampling [11] and voxelization tricks [25, 46] to further improve the inference speed. Alternatively, one could consider studying compound scaling, like that in EfficientNet [36].

References

[1] Iro Armeni, O. Sener, A. Zamir, Helen Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1534–1543, 2016.

[2] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 3(4):3145–3152, 2018.

[3] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[4] François Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[5] Y. Feng, Zizhao Zhang, X. Zhao, R. Ji, and Yue Gao. Gvcnn: Group-view convolutional neural networks for 3d shape recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 264–272, 2018.

[6] B. Graham, Martin Engelcke, and L. V. D. Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018.

[7] Fabian Groh, Patrick Wieschollek, and Hendrik P. A. Lensch. Flex-convolution (million-scale point-cloud learning beyond grid-worlds). In *Asian Conference on Computer Vision (ACCV)*, Dezember 2018.

[8] P. Guerrero, Yanir Kleiman, M. Ovsjanikov, and N. Mitra. Pcpnet learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37, 2018.

[9] William L. Hamilton, Zhitao Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[10] A. Howard, Menglong Zhu, Bo Chen, D. Kalenichenko, W. Wang, Tobias Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

[11] Q. Hu, B. Yang, Linhai Xie, S. Rosa, Yulan Guo, Zhihua Wang, A. Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11105–11114, 2020.

[12] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*. OpenReview.net, 2017.

[16] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018.

[17] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018.

[18] Eric-Tuan Le, I. Kokkinos, and N. Mitra. Going deeper with lean point networks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9500–9509, 2020.

[19] Huan Lei, Naveed Akhtar, and A. Mian. Seggcn: Efficient 3d point cloud segmentation with fuzzy spherical kernel. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11608–11617, 2020.

[20] G. Li, M. Müller, Guocheng Qian, Itzel C. Delgadillo, Abdulellah Abualshour, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Making gcns go as deep as cnns. *IEEE transactions on pattern analysis and machine intelligence*, PP, 2021.

[21] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[22] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. In *NeurIPS*, 2018.

[23] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8887–8896, 2019.

[24] Z. Liu, H. Hu, Yue Cao, Zheng Zhang, and Xin Tong. A closer look at local aggregation operators in point cloud analysis. In *ECCV*, 2020.

[25] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. In *NeurIPS*, 2019.

[26] Ningning Ma, X. Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[27] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3d point cloud understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1578–1587, 2019.

[28] Charles R. Qi, O. Litany, Kaiming He, and L. Guibas. Deep hough voting for 3d object detection in point clouds. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9276–9285, 2019.

[29] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[30] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, pages 5099–5108, 2017.

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI (3)*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.

[32] M. Sandler, Andrew G. Howard, Menglong Zhu, A. Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[33] Shaoshuai Shi, X. Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–779, 2019.

[34] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[35] Hang Su, Subhransu Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 945–953, 2015.

[36] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.

[37] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, J. Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *ECCV*, 2020.

[38] Maxim Tatarchenko, Jaesik Park, V. Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018.

[39] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6410–6419, 2019.

[40] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10296–10305, 2019.

[41] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018.

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

[43] Xin Wei, Ruixuan Yu, and J. Sun. View-gcn: View-based graph convolutional network for 3d shape analysis. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1847–1856, 2020.

[44] B. Wu, Alvin Wan, Xiangyu Yue, P. Jin, S. Zhao, Noah Golmant, A. Gholaminejad, Joseph E. Gonzalez, and K. Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9127–9135, 2018.

[45] Wenxuan Wu, Zhongang Qi, and F. Li. Pointconv: Deep convolutional networks on 3d point clouds. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9613–9622, 2019.

[46] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and U. Neumann. Grid-gcn for fast and scalable point cloud learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5660–5669, 2020.

[47] Yifan Xu, Tianqi Fan, Mingye Xu, L. Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *ECCV*, 2018.

[48] Yan Yan, Yuxing Mao, and B. Li. Second: Sparsely embedded convolutional detection. *Sensors (Basel, Switzerland)*, 18, 2018.

[49] Chris Zhang, Wenjie Luo, and Raquel Urtasun. Efficient convolutions for real-time semantic segmentation of 3d point clouds. In *2018 International Conference on 3D Vision (3DV)*, pages 399–408. IEEE, 2018.

[50] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1607–1616, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] refer to Sections 6
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Available in the supplementary material.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Available in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] We train all the experiments using seed 0 in a deterministic setting like the state-of-the-art [39, 24].

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] All of the codes are publicly available and are mostly under MIT public license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All the data is publicly available.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]