PointGL: A Simple Global-Local Framework for Efficient Point Cloud Analysis

Jianan Li^{*†}, Jie Wang^{*}, and Tingfa Xu[†]

Abstract-Efficient analysis of point clouds holds paramount significance in real-world 3D applications. Currently, prevailing point-based models adhere to the PointNet++ methodology, which involves embedding and abstracting point features within a sequence of spatially overlapping local point sets, resulting in noticeable computational redundancy. Drawing inspiration from the streamlined paradigm of pixel embedding followed by regional pooling in Convolutional Neural Networks (CNNs), we introduce a novel, uncomplicated yet potent architecture known as PointGL, crafted to facilitate efficient point cloud analysis. PointGL employs a hierarchical process of feature acquisition through two recursive steps. First, the Global Point Embedding leverages straightforward residual Multilayer Perceptrons (MLPs) to effectuate feature embedding for each individual point. Second, the novel Local Graph Pooling technique characterizes point-to-point relationships and abstracts regional representations through succinct local graphs. The harmonious fusion of one-time point embedding and parameter-free graph pooling contributes to PointGL's defining attributes of minimized model complexity and heightened efficiency. Our PointGL attains stateof-the-art accuracy on the ScanObjectNN dataset while exhibiting a runtime that is more than 5 times faster and utilizing only approximately 4% of the FLOPs and 30% of the parameters compared to the recent PointMLP model. The code for PointGL is available at https://github.com/Roywangj/PointGL.

Index Terms-Point cloud, feature embedding, graph

I. INTRODUCTION

Acquiring concise geometric features from point clouds stands as a pivotal stride across a spectrum of 3D tasks [1], [2], [3] and multimedia applications [4], [5], [6], [7]. Amid the gamut of methods for point cloud analysis, point-based models have been exhaustively investigated owing to their adept equilibrium between precision and efficiency. Noteworthy among these is PointNet++ [8], which is hailed as the progenitor in this domain. Although subsequent endeavors have predominantly fixated on enhancing precision, these enhancements frequently exact a toll in terms of augmented computational intricacy. Hence, our study strives to fashion a point-based model that amalgamates simplicity and potency, thereby facilitating expeditious analysis of point clouds.

Contemporary point-based models conventionally embrace the PointNet++ pipeline. This pipeline encompasses the grouping of input points into local sets and subsequently executes



Fig. 1. Comparing Accuracy and Speed Among Various Approaches on the ScanObjectNN Dataset. Our PointGL achieves the highest predictive accuracy and exhibits the fastest inference speed compared to alternative approaches.

point-wise embedding procedures within these local point sets. The ensuing phase entails employing symmetrical aggregation functions to distill local geometric features within each distinct point set. Nonetheless, a comprehensive scrutiny of this pipeline has brought to the fore two intrinsic drawbacks that curtail computational efficiency.

The initial inadequacy we discerned pertains to the substantial overlapping of the resultant local point sets acquired through grouping within the 3D space. This overlap results in a singular input point being encompassed by multiple point sets concurrently. Given that the embedding of points is executed individually within each local point set, this scenario leads to the replication of point embeddings for the same point across diverse point sets. Although minor disparities may arise in these embeddings due to variances in the relative positions of input points, we posit that this process entails noteworthy redundant computations.

The second inherent flaw is interconnected with the constraints posed by the process of extracting local features via point-wise embedding, succeeded by the application of symmetric functions like max pooling. This methodology can potentially lead to the erosion of intricate geometry due to its relatively feeble depiction of point-to-point relationships. Despite subsequent endeavors aimed at ameliorating this concern through the integration of more intricate encoders, such as convolutional schemes [9], [10], graph-based techniques [11], or self-attention mechanisms [12], [13], these enhancements frequently entail a marked escalation in computational expenditure. As a result, it becomes imperative to identify more

^{*}Equal contribution. [†] Correspondence to: Jianan Li and Tingfa Xu.

J. Li, J. Wang and T. Xu are with Beijing Institute of Technology, Beijing 100081, China {lijianan,ciom_xtf1}@bit.edu.cn

J. Li and T. Xu are also with the Key Laboratory of Photoelectronic Imaging Technology and System, Ministry of Education of China, Beijing 100081, China.

T. Xu is also with Chongqing Innovation Center, Beijing Institute of Technology, Chongqing 401135, China.

efficient approaches for the delineation and amalgamation of point-to-point relationships while upholding a commendable level of precision.

The abovementioned constraints impel us to explore a novel, efficient point-based paradigm guided by a concise design philosophy. Our design is grounded in a pivotal insight that individualized per-point feature embedding may not be indispensably correlated with optimal performance, and may inadvertently give rise to superfluous computational efforts. To surmount this, we execute feature embedding for each input point prior to the grouping of sets. This approach entails a singular embedding operation per point, engendering notable computational economies. Nonetheless, a pivotal quandary arises from the isolation of feature embedding for each point, consequently overlooking the interdependencies amongst points. These interrelationships, however, play a momentous role in capturing spatial geometric attributes.

In light of this, we introduce an additional step subsequent to the per-point embedding process, which intricately interlinks points within a designated local region and encapsulates their correlations within the local representation. To accomplish this task while preserving efficiency, we advocate for the incorporation of a graph structure, as it is adeptly suited for delineating interactions among nodes (points), thereby ensuring efficiency by streamlining the computation of edge features (relations).

Building upon the previously elucidated concepts, we have devised a novel architecture termed PointGL, tailored for the efficient analysis of point clouds. PointGL elucidates hierarchical features from the input point cloud via the integration of multiple progressive learning stages. Each of these stages encompasses two distinct phases: a Global Point Embedding phase and a Local Graph Pooling phase. The former assumes responsibility for the embedding of features per point, achieved by characterizing each point through elementary residual multi-layer perceptron (MLP) blocks. On the other hand, the latter encapsulates regional features by establishing connections amongst points within a designated local region, effectively constituting a rudimentary graph. To ensure streamlined operations, we endow the graph edges with difference features derived from the disparities between the regional center point and its adjoining neighbors. Remarkably, our investigations have unveiled the remarkable informativeness of these edge features in delineating regional point-to-point relationships. Subsequently, these features are amalgamated via straightforward max pooling techniques, culminating in the extraction of regional representations.

Significantly, the process of local graph pooling is distinguished by its concise and efficient workflow, requiring minimal introduction of additional parameters. This inherent efficiency stems from the meticulous orchestration of global point embedding and local graph pooling, culminating in a streamlined and proficient framework for analysing point clouds. This framework not only excels in delivering high predictive accuracy but also exhibits notable swiftness in computational execution.

We have undertaken an extensive evaluation of PointGL across multiple benchmark datasets. As illustrated in Fig. 1,

our PointGL framework attains a state-of-the-art level of accuracy while concurrently delivering remarkable efficiency in inference speed. Noteworthy is the observation that our approach executes computations at nearly the same pace as PointNet++, yet remarkably enhances the overall accuracy by 9%. Additionally, PointGL achieves accuracy levels comparable to those of PointMLP [14], yet operates more than 5 times faster, utilizing only about 4% of the Floating-Point Operations (FLOPs) and 30% of the parameters.

Moreover, we have extended our assessments to encompass segmentation and object detection, reaffirming the efficacy of PointGL in diverse downstream tasks. Furthermore, PointGL consistently showcases robustness in the face of point cloud corruptions. This resilience is substantiated by its attainment of state-of-the-art results across both the ModelNet-C and ShapeNet-C benchmarks, thereby solidifying its potential as a promising contender for real-world applications.

In summary, this study contributes in the following ways:

- We introduce PointGL, which pioneers a novel and efficient point-based paradigm tailored for point cloud analysis. Despite its straightforward design, PointGL emerges as a potent tool for analyzing point clouds, thereby enriching the repertoire of existing point-based models.
- A novel local graph pooling operation is introduced, enabling the streamlined extraction of local geometric features. Notably, this operation demands minimal parameterization and seamless integration into pre-existing models.
- PointGL emerges as a lightweight model characterized by diminished computational intricacy. This attribute strikes an optimal equilibrium between precision and efficiency. Furthermore, its pronounced resilience in the face of corruptions reinforces its viability for real-world applications.

II. RELATED WORK

Learning on Point Clouds. The acquisition of discriminative features from point clouds constitutes a foundational endeavor for diverse 3D vision applications. Nonetheless, the inherent irregularity within point cloud data poses a challenge to conventional methodologies, such as voxel-based techniques [15], [16], and view-based approaches [17], [18], [19], [20], [21]. The former method involves projecting point clouds onto structured voxel grids, while the latter transforms them into 2D images. Both methodologies aim to capitalize on solutions developed for structured data, yet they suffer from a reduction of information due to the inherent projection process. In contrast, point-based techniques [22], [8], [11] tackle this issue by directly processing the raw point cloud data. PointNet [22], a pivotal advancement in 3D comprehension, employs multiple shared multi-layer perceptrons (MLPs) to learn individual point-wise features. This is achieved while maintaining permutation invariance through a max-pooling operator. Its successor, PointNet++ [8], enhances this approach by extracting both global and local geometric information through MLPs and a hierarchical architecture. This enables

the extraction of more robust representations of point clouds. In this paper, we propose a more efficient and simplified hierarchical learning paradigm designed to capture the local geometric features of point clouds.

Extraction of Local Geometric Features. The identification of local geometric features holds significant importance within the realm of point cloud analysis, a notion underscored by Liu et al. [23]. Previous investigations in this domain [8], [11] have introduced diverse methodologies to address this challenge. For instance, PointNet++ [8] has employed shared MLPs coupled with max pooling, while DGCNN [11] introduced EdgeConv to capture interpoint relationships and DC-GNN [24] incorporates a channel dropout mechanism alongside a hierarchical feature selection strategy at each network layer for dynamic graph construction. Alternative strategies [25], [13] have explored deformable kernels, as exemplified by KPConv [25], or integrated attention mechanisms, as showcased in Point Transformer [13], to model point interactions. Furthermore, PointMLP [14] utilizes a straightforward feed-forward residual MLP module accompanied by a geometric affine module to extract local features. This serves to demonstrate that a simplistic hierarchical MLP architecture can yield commendable performance while sidestepping the need for intricate local geometric extractors. In a similar vein, PointNext [26] employs an Inverted Residual MLP to abstract local features, yielding favorable results. Within this study, we propose a parsimonious yet efficacious local graph pooling approach, which manifests exceptional prowess in abstracting local features. Notably, this comes at the expense of minimal supplementary parameters and computational overhead.

Architectural Frameworks for Point Clouds. Within the realm of point cloud analysis, various architectural frameworks have emerged to process point cloud data. These include MLP-based approaches [22], [8], [14], convolution-based methods [27], [28], [9], graph-based models [11], [29], relation-based models [30], and transformer-based models [12], [13]. While these architectures have demonstrated impressive accuracy, they often suffer from protracted inference times. In contrast, this paper advocates for a concise design philosophy, eschewing convoluted architectures in favor of presenting a streamlined yet potent framework for efficient point cloud analysis. This approach endeavors to strike a harmonious balance between accuracy and efficiency, a facet of particular significance in real-world applications.

Robustness of Point Cloud Models. The assessment of the robustness of deep learning models is imperative prior to their deployment in real-world scenarios. In recent times, the research community has directed its efforts toward formulating benchmarks to gauge the resilience of models using 2D image datasets. Examples include ImageNet-C [31], Imagenet-V2 [32], and ObjectNet [33]. Concerning 3D point cloud data, Ren et al. [34] have introduced a classification scheme for common 3D corruptions and subsequently conducted an extensive evaluation of extant point cloud classification models. Their findings underscore the susceptibility of prevailing point cloud models to corruptions. In contrast, the uncomplicated architectural framework advocated in this study exhibits commendable robustness in the presence of corruptions.

III. METHOD

A. Revisiting Point-based Approaches

Point-based methods, notably exemplified by PointNet [22] and PointNet++ [8], stand as pioneering instances in this domain, furnishing a foundational framework for subsequent endeavors. PointNet++ undertakes the hierarchical abstraction of intrinsic geometric features across multiple stages, facilitated by a collection of N points, each characterized by Cartesian coordinates denoted as $\mathcal{P} = \{p_i \in \mathbb{R}^{1\times 3} | i = 1, \dots, N\}$. Within the *s*-th stage, PointNet++ employs the farthest point sampling (FPS) algorithm to select N_s points. For each of these sampled points, K neighbors are queried, thereby constituting N_s local point sets. Subsequently, PointNet++ proceeds to acquire insights into the local patterns within each local point set through the expression:

$$\boldsymbol{g}_{i} = \mathcal{A}_{j=1,\cdots,\mathrm{K}} \left\{ \boldsymbol{h}\left(\boldsymbol{f}_{i,j}\right) \right\},$$
 (1)

Here, $f_{i,j}$ signifies the feature of the *j*-th neighboring point pertaining to the *i*-th sampled point and g_i denotes the aggregated local feature for the *i*-th sampled point. The function $h(\cdot)$ is implemented by a multi-layer perceptron (MLP), thus conducting spatial encoding for a given point. Meanwhile, \mathcal{A} represents a symmetric function, like max pooling, employed to consolidate the encoded point features. Through the aggregation of multiple stages, featuring diminishing N_s and augmented spatial coverage within each local point set, PointNet++ progressively engenders abstracted local geometric features coupled with an expanding receptive field.

Subsequent point-based methodologies [10], [35], [25], [13], [36] have primarily directed their efforts towards enhancing the effectiveness of $h(\cdot)$ to more adeptly capture interrelations between points, achieved by integrating convolution, graph, or attention mechanisms. Notably, the majority of these methodologies adhere to the overarching framework established by PointNet++: during each stage, input points are conglomerated into a sequence of local point sets, wherein feature embedding is executed for the points within each point set. Subsequently, local features are distilled through an aggregation function. However, a comprehensive scrutiny of this framework has illuminated two inherent limitations that could potentially impede computational efficiency.

Primarily, it is evident that during the *s*-th stage, there are N_{s-1} input points. Following the process of grouping, N_s local point sets emerge, with each assemblage accommodating K neighboring points. Typically, N_s constitutes merely half of N_{s-1} , and K is selected from the range of [16, 64], thus implying that the majority of input points are represented in multiple point sets. Given that $h(\cdot)$ operates on every point within each point set, on average, every input point necessitates a cumulative count of $N_s K/N_{s-1}$ feature embeddings. While point embeddings within distinct point sets might encompass varying spatial details, such as relative coordinates concerning a point set center, we posit that subjecting an individual point to multiple feature embeddings might not be pivotal for performance enhancement. However, this approach unavoidably introduces a substantial degree of computational



Fig. 2. Comprehensive Architecture of PointGL. PointGL's architecture involves the extraction of hierarchical features from input point clouds through the stacking of multiple learning stages. Each stage initiates with a global point embedding phase, wherein feature embedding for individual points is conducted using residual MLP blocks. Following this, a local graph pooling phase captures and abstracts point-to-point relations into local representations by constructing a concise regional graph centered around each sampled point. The synergistic fusion of global point embedding and local graph pooling culminates in a coherent and efficient hierarchical framework for point cloud analysis.

superfluity. It's paramount to recognize that this impact becomes significantly magnified upon the integration of intricate spatial encoders in lieu of conventional MLPs.

Secondly, a noteworthy observation pertains to the fact that PointNet++ embarks on the abstraction of geometric features within a local point set via point-wise MLP, succeeded by a symmetric function. Regrettably, this approach is prone to diluting intricate geometric subtleties, primarily stemming from its limited capacity to model interpoint relationships. This shortcoming has prompted subsequent studies to address the concern through the introduction of intricate spatial encoders or aggregation functions. Yet, the unintended consequence of these endeavors has been the imposition of exorbitant computational demands and a significant expansion in memory usage, thus compromising overall efficiency. Hence, the endeavor to enhance efficiency while upholding accuracy necessitates a more streamlined approach to the modeling and aggregation of interpoint relationships. These two limitations collectively impel us to embark on an exploration of a novel point-based paradigm.

B. PointGL Framework

The comprehensive architecture of our PointGL is elucidated in Fig. 2, detailing its process of learning hierarchical features from input point clouds through the stacking of a total of S learning stages. In the context of the *s*-th stage, the input is constituted by N_{s-1} points denoted as $\mathcal{P}_{s-1} = \{(\mathbf{p}_i, \mathbf{f}_i) \mid i = 1, \dots, N_{s-1}\}$. Here, each point *i* is characterized by its *xyz* Cartesian coordinates represented as $\mathbf{p}_i \in \mathbb{R}^3$, along with an associated feature vector $\mathbf{f}_i \in \mathbb{R}^{D_{s-1}}$. The outcome of this stage encompasses N_s sampled points denoted by $\mathcal{P}_s = \{(\mathbf{p}_i, \mathbf{g}_i) \mid i = 1, \dots, N_s\}$. Within this set, each sampled point *i* is distinguished by a feature vector $\mathbf{g}_i \in \mathbb{R}^{D_s}$ that encapsulates the localized pattern surrounding it.

Our PointGL introduces a novel and efficient approach to learn local patterns through a dual-step process. The first step, known as Global Point Embedding, entails the execution of feature embedding for each point within the input set \mathcal{P}_{s-1} . Diverging from the methodology of PointNet++, which generates spatially overlapping local point sets and rigorously encodes features for each point within each point set, PointGL undertakes feature embedding solely once for every input point. This strategic divergence substantially mitigates redundant computations. Subsequently, in the subsequent step known as Local Graph Pooling, points are selectively sampled from \mathcal{P}_{s-1} , and the relationships between each sampled point and its neighboring points are meticulously modeled and collectively integrated to formulate localized representations. Remarkably, this entire sequence is accomplished exclusively through straightforward operations, effectively upholding computational efficiency. In the following sections, we proceed to provide an in-depth elaboration of both of these fundamental steps.

Global Point Embedding. This phase entails the embedding of each input point within \mathcal{P}_{s-1} by training a function Φ : $\mathbb{R}^{D_{s-1}} \to \mathbb{R}^{D_s}$ dedicated to feature embedding. The feature embedding process is as follows:

$$\boldsymbol{v}_{i} = \boldsymbol{\Phi}\left(\boldsymbol{f}_{i}\right), i = 1, \cdots, N_{s-1},$$
 (2)

where v_i is the embedded feature for the *i*-th point. Following the approach introduced in [14], we adopt the perspective of $\Phi(\cdot)$ as a sequence of residual point MLP blocks. Precisely, this function is learned through a sequence of uniform residual MLP blocks. The composition of MLP encompasses a fully connected (FC) layer, batch normalization (BN), and rectified linear unit (ReLU) activation, succeeded by another layer of FC and BN.

The employment of point-wise MLPs imparts invariance to point permutations upon the function $\Phi(\cdot)$, which is capable of

approximating a diverse array of continuous functions. Additionally, the point embedding procedure necessitates minimal operations, primarily reliant on meticulously optimized feedforward MLPs. Furthermore, given that point embedding is executed solely once for each input point, the efficiency of our PointGL remains unimpeded even with the integration of more intricate or deeper MLPs.

Local Graph Pooling. The objective of this phase is to derive local patterns through the modeling and aggregation of interpoint relationships within a confined area. Recognizing the efficacy of graphs in representing sets of objects (nodes) and their interconnections (edges), a logical approach to capture point-to-point relationships within a limited region is to establish a local graph by connecting internal points. Within this construct, each node corresponds to a point, and the edges symbolize interactions between pairs of points. Through the amalgamation of all edge features within the local graph, we can effectively unearth local patterns.

To achieve this, we undertake the process through three distinct steps of local point association followed by feature aggregation: *i*) We initiate the process by sampling N_s points from the input data at the current stage utilizing the farthest point sampling (FPS) technique. Following this, we employ the k-nearest neighbors (kNN) algorithm to gather K neighboring points corresponding to each of the sampled points. *ii*) Subsequently, each sampled point is connected to its respective neighbors, effectively constituting a series of local graphs. These graphs encode the point-to-point relations through the incorporation of edge features. *iii*) In the final step, we subject the edge features of each local graph to a symmetric aggregation function. This operation yields the local feature representation.

The pivotal stride involves generating edge features capable of effectively encoding relationships among points while upholding efficiency. To accomplish this, we embrace a rudimentary yet remarkably efficacious strategy, encompassing the computation of the difference feature between a sampled point and each of its neighbors as the edge feature:

$$\boldsymbol{e}_{j,k} = \boldsymbol{\alpha} \odot (\boldsymbol{v}_{j,k} - \boldsymbol{v}_j) + \boldsymbol{\beta}. \tag{3}$$

In this context, $v_j \in \mathbb{R}^{D_s}$ and $v_{j,k} \in \mathbb{R}^{D_s}$ signify the features of the *j*-th sampled point and its *k*-th neighboring point, respectively. The parameters $\alpha, \beta \in \mathbb{R}^{D_s}$ are subject to learning, with the initial value of 1 and 0, respectively, while \odot signifies the Hadamard product.

The local output representation for the *j*-th sampled point, denoted as g_j , is derived by aggregating the edge features linked with all the edges emanating from this point. This aggregation is defined as:

$$\boldsymbol{g}_{j} = \boldsymbol{\mathcal{A}}(\boldsymbol{e}_{j,k} | k = 1, \cdots, \mathbf{K}), \qquad (4)$$

where K is the number of neighboring points. We opt for the employment of the max pooling operation as the aggregation function $\mathcal{A}(\cdot)$ due to its efficacy and simplicity.

Beyond its simplicity and compactness, our approach to local graph pooling offers several salient advantages: i) it retains invariance to the ordering of neighbors, thereby ensuring resistance to variations in point permutations; ii) it refrains

```
Algorithm 1 Pseudo code of PointGL in a Pytorch-like style.
Input:
 xyz - [N, 3], coordinate of the input point cloud, where N
       denotes the number of points
 points - [N, C], feature of the input point cloud, where C
       denotes the dimension of the feature
 ns - number of the points selected by Farthest Point
      Sampling (FPS) algorithm
 K - number of the neighboring points by k-nearest neighbor
        (kNN) algorithm
 lpha - affine transformation parameter, default 1.0
 \beta - affine transformation parameter, default 0.0
Output:
 new_xyz - [ns, 3], coordinate of the output point cloud,
      where ns denotes the number of points
 new_points - [ns, C'], feature of the output point cloud,
where C' denotes the dimension of the feature
Function local_graph_pooling(xyz, points, ns, K, \alpha, \beta):
   # Select points by FPS algorithm
   fps_idx = furthest_point_sample(xyz, ns)
   new_xyz = gather(xyz, fps_idx)
   new_points = gather(points, fps_idx)
   # Find neighboring points by k-NN algorithm
   grouped_xyz, grouped_points = knn(query_xyz=new_xyz,
        support_xyz=xyz, feat=points, k_number=K)
   # Generate edge features
   grouped_points = grouped_points - new_points
   grouped_points = \alpha * \text{grouped_points} + \beta
   # Aggregate local features
   new_points = grouped_points.max[0]
   return new_xyz, new_points
In Each Processing Stage:
   # Step1: Global Point Embedding
   points = residual_MLPs(points)
   # Step2: Local Graph Pooling
   new_xyz, new_points = local_graph_pooling(xyz, points, ns
        , K, \alpha, \beta)
```

from involving intricate operations, thus ensuring commendable computational efficiency; *iii*) it is nearly parameter-free, facilitating its seamless integration in a plug-and-play manner.

C. Architectural Details

As elucidated earlier, each learning stage within our methodology samples a subset of points from the input, endowing each sampled point with local geometric information. Through the accumulation of multiple stages, this design progressively furnishes a smaller number of points, each enriched with an extended contextual awareness.

The PointGL network is structured with S = 4 stages. These stages exhibit escalating embedding dimensions of $D_s = \{128, 256, 512, 1024\}$, complemented by a diminishing count of sampled points denoted as $Ns = \{512, 256, 128, 64\}$. Each stage encompasses a residual MLP block, leveraging K = 24 nearest neighbors for the purpose of local feature aggregation.

To further optimize efficiency, drawing inspiration from [14], we have introduced an advanced iteration of PointGL, denoted as PointGL-elite. This variant refines the feature embedding dimensions based on the original PointGL framework. For a comprehensive overview of the architectural nuances, please consult Table I. In addition, Algorithm 1 shows the implementation of PointGL.
 TABLE I

 ARCHITECTURAL SPECIFICATIONS. MLPS SPECIFICATION: OUTPUT CHANNEL COUNT OF FULLY CONNECTED (FC) LAYERS. POOLING SPECIFICATION: (Ns: NUMBER OF SAMPLED POINTS, K: NUMBER OF NEIGHBORING POINTS).

Model		Poi	intGL	PointGL-elite				
Stage	S_1	S ₂	S ₃	S_4	S_1	S_2	S ₃	S_4
	[128]	[256]	[512]	[1024]	[64]	[128]	[256]	[256]
MLPs	[128, 128]	[256, 256]	[512, 512]	[1024, 1024]	[16, 64]	[32, 128]	[64, 256]	[64, 256]
	[128, 128]	[256, 256]	[512, 512]	[1024, 1024]				
Pooling	(512, 24)	(256, 24)	(128, 24)	(64, 24)	(512, 24)	(256, 24)	(128, 24)	(64, 24)
Output	512×128	256×256	128×512	64×1024	512×64	256×128	128×256	64×256

TABLE II

PERFORMANCE COMPARISON ON SCANOBJECTNN AND MODELNET40 DATASETS. THE TABLE PRESENTS METRICS INCLUDING OVERALL ACCURACY (OA, %); MEAN PER-CLASS ACCURACY (MACC, %); PARAMETER COUNT (#PARAMS); AND FLOPS. ADDITIONALLY, WE ASSESS THE PROCESSING SPEED OF ALL METHODS IN SAMPLES PER SECOND ON A SINGLE GEFORCE RTX 3090 GPU AND TWO CORES OF AN INTEL XEON GOLD 5218R CPU@2.10GHz. (†: MULTI-SCALE INFERENCE AS PER [35].)

Method	Input	Input ScanObjectNN		Mode	elNet40	#Params	FI OPc	Train	Infer
Wiethou	Input	OA(%)	mAcc(%)	OA (%)	mAcc(%)		FLOI S	Speed	Speed
PointNet [22]	1k P	68.2	63.4	89.2	86.0	3.47M	0.45G	1104.3	2029.6
DGCNN [11]	1k P	78.1	73.6	92.9	90.2	1.82M	2.43G	275.4	517.7
KPConv [25]	7k P	-	-	92.9	-	14.30M	-	152.8	309.9
ASSANet (L) [37]	1k P	80.8	77.7	92.9	-	-	2.72G	312.8	926.7
PointASNL [38]	1k P*	-	-	93.2	-	10.1M	1.80G	-	-
MVTN [39]	multi-view	82.8	-	93.8	92.0	4.24M	1.78G	-	-
PAConv [†] [10]	1k P	-	-	93.9	-	2.44M	1.68G	-	-
RPNet [30]	1k P*	-	-	94.1	-	2.70M	3.90G	-	-
CurveNet [40]	1k P	-	-	93.8	-	2.14M	0.66G	126.8	278.9
PointNet++ [8]	1k P	77.9	75.4	90.7	88.4	1.48M	0.86G	693.6	1473.4
PointMLP [14]	1k P	85.4	83.9	94.1	91.3	13.24M	15.67G	88.4	230.2
PointMLP [†] [14]	1k P	86.5	85.1	94.5	91.4	13.24M	15.67G	88.4	230.2
PointMLP-elite [14]	1k P	83.8	81.8	93.6	90.9	0.72M	0.91G	324.9	681.1
RepSurf-U [41]	1k P	84.3	81.3	94.4	91.4	1.48M	0.90G	73.4	377.2
RepSurf-U [†] [41]	1k P	84.6	81.9	94.7	91.7	1.48M	0.90G	73.4	377.2
PointGL	1k P	86.9	85.2	93.0	90.4	4.16M	0.63G	600.7	1395.6
PointGL-elite	1k P	84.2	82.2	92.6	89.8	0.49M	0.05G	1103.5	2035.3

IV. EXPERIMENT

A comprehensive evaluation of the PointGL approach was undertaken across diverse benchmarks to gauge its effectiveness. Furthermore, we conducted experiments to evaluate the method's robustness in the face of corruptions, and performed ablation studies to validate both the chosen design principles and parameter configurations.

A. Shape Classification on ScanObjectNN

Data and Setup. The fundamental assessment of point cloud analysis methods relies on 3D object classification. For our primary evaluation, we utilized the ScanObjectNN benchmark [42]. This benchmark draws upon real-world object instances to provide an authentic assessment of the model's performance. The benchmark constitutes a multi-class classification task, which comprises a total of 2,902 real-world point clouds from across 15 distinct classes.

During the training phase, both the PointGL and PointGLelite models underwent training for 250 epochs utilizing the AdamW optimizer with a batch size of 32. The optimizer utilizes a learning rate of 0.002 and weight decay of 0.05. The CosineAnnealingLR [43] scheduler is employed to decrease the learning rate to the minimum value of 1e-4, and the warm up epochs is set to 0. The evaluation metrics encompass overall accuracy (OA) and class-average accuracy (mAcc) calculated on the test set.

Main Results. The Table II furnishes a comparative analysis between PointGL and state-of-the-art techniques. To ensure a comprehensive evaluation of the various methods, an assortment of metrics was considered, encompassing accuracy and efficiency measurements such as classification accuracy, model complexity (parameter count and FLOPs), and processing speed. Notably, our PointGL exhibited superior levels of accuracy and efficiency within the purview of this benchmark assessment.

In a specific context, PointGL showcased a cutting-edge overall accuracy of 86.9%, asserting its dominance over preceding methodologies such as PointMLP [14] and Rep-Surf [41]. Moreover, our self-contained PointGL demonstrated superiority over both PointMLP and RepSurf, even when employing a multi-scale inference strategy [35]. Of significant note is our approach's substantial performance advantage

 TABLE III

 Ablations on **Global point embedding** and **Local graph pooling** on the ScanObjectNN dataset.

Position		A(%)	mAcc(%)	#Params	FLOPs	Train	Infer	Depth	OA(%)	mAcc(%)
		-(,-)				Speed	Speed	16 layers	86.9	85.2
Pos-kNN		36.2	84.0	4.16M	7.55G	222.1	563.8	24 layers	86.6 86.4	84.6 84.5
Pre-kNN (re-kNN (Ours) 86.9		85.2	4.16M	0.63G	600.7	1395.6	32 layers		
		a) Perform	n point embed	ding before/af	ter kNN.			(b) Number	of embedding	g layers.
Max Pool	Local Grap	a) Perform n α - β	n point embed	ding before/af	ter kNN.	OA(%)	mAcc(%)	(b) Number Model	of embedding	g layers. mAcc(%)
Max Pool	Local Grap	a) Perform $\alpha - \beta$	OA(%)	mAcc(%) 80.3	$\frac{1}{\frac{K}{12}}$	OA(%) 85.8	mAcc(%) 83.7	(b) Number Model PointNet++	of embedding OA(%) 77.9	g layers. mAcc(%) 75.4
Max Pool	Local Grap	a) Perform $\alpha - \beta$ χ χ	a point embed OA(%) 82.1 86.1 ↑4.0	ding before/af mAcc(%) 80.3 84.5 ↑4.2	$\frac{K}{12}$	OA(%) 85.8 86.9	mAcc(%) 83.7 85.2	(b) Number Model PointNet++ PointNet++ (LGP)	of embedding OA(%) 77.9 78.7 ↑0.8	g layers. mAcc(%) 75.4 75.7 ↑0.3
Max Pool	Local Grap	a) Perform $\alpha - \beta$ χ χ	n point embed OA(%) 82.1 86.1 ↑4.0 86.9 ↑0.8	mAcc(%) 80.3 84.5 ↑4.2 85.2 ↑0.7	K I 12 24 36 I	OA(%) 85.8 86.9 86.4	mAcc(%) 83.7 85.2 85.0	(b) Number Model PointNet++ PointNet++ (LGP) PointMLP	of embedding OA(%) 77.9 78.7 ↑0.8 85.4	mAcc(%) 75.4 75.7 ↑0.3 83.9

(c) Component ablation.

(d) Number of neighboring points.

(e) Plug-and-play capability.

over PointNet++ by a substantial margin (86.9% vs. 77.9%), thereby underscoring the supremacy of our local graph pooling mechanism for the abstraction of local patterns.

Computational Efficiency. Regarding computational efficiency, our PointGL operates at a pace that is nearly comparable to PointNet++, widely acknowledged as the swiftest pointbased model with a hierarchical framework. When contrasted with the recent PointMLP, our PointGL boasts a mere **30**% of the parameter count (4.16M *vs.* 13.24M) and a mere **4**% of the FLOPs (0.63G *vs.* 15.67G), all while delivering over **5**× the speed in inference (1, 395 *vs.* 230 samples/s).

In contrast to the more contemporary RepSurf-U, our PointGL experiences a slight increase in FLOPs, yet manages to achieve $7.2 \times$ and $2.7 \times$ faster speeds during training and inference, respectively. These findings underscore that a reduction in model complexity doesn't necessarily guarantee heightened efficiency. Our straightforward approach of global point embedding followed by local graph pooling effectively curtails redundant computations across local point sets, thereby serving as the fundamental basis for efficiency enhancement.

To further optimize efficiency, we introduce an accelerated version of PointGL termed PointGL-elite. With a mere 0.49M parameters, PointGL-elite significantly reduces FLOPs to just 0.05G, accounting for only 5% of the FLOPs found in its PointMLP-elite counterpart (0.91G). Despite maintaining a competitive accuracy level of 84.2% OA, PointGL-elite achieves an exceptional inference speed of 2,035 samples/s, surpassing the speeds of PointMLP-elite and PointNet++ by nearly $3 \times$ and $1.4 \times$, respectively. These outcomes firmly establish PointGL-elite as the point-based model with minimal model complexity and the swiftest inference rate.

B. Shape Classification on ModelNet40

Data and Experimental Setup. Furthermore, we conducted evaluations using the ModelNet40 benchmark [44], which is a multi-class classification task that encompasses a collection of 9,843 training and 2,468 test CAD models, distributed across 40 distinct categories. The training process for both PointGL and PointGL-elite involved employing the Stochastic Gradient Descent (SGD) optimizer over 300 epochs with a batch size of 32. The optimizer is configured with a learning rate of 0.1, momentum of 0.9 and weight decay of 2e-4. The

learning rate is decayed to the minimum value of 0.005 using the CosineLRScheduler scheme.

Main Findings and Results. The comprehensive comparison with leading state-of-the-art methodologies is presented in Table II. In the absence of utilizing a voting mechanism, our PointGL and PointGL-elite attain impressive overall accuracies of 93.0% and 92.6%, respectively, notably surpassing PointNet++ by a considerable margin of 2.3% and 1.9%. Although exhibiting a slight decrease in accuracy compared to CurveNet, our approach showcases substantial reductions in model complexity while simultaneously achieving enhanced efficiency. Specifically, PointGL-elite stands out with just 23% of the parameters and 8% of the FLOPs, while exhibiting training and inference speeds that are accelerated by 7.7-fold and 6.3-fold, respectively.

C. Ablation Studies

Global Point Embedding. To validate our key observation that conducting per-point feature embedding within each local point set, akin to the PointNet++ approach, is unnecessary for achieving optimal performance and would introduce significant computational redundancy, we devised a model variant. This variant involves relocating the point embedding step to the local graph pooling stage, positioned immediately after the k-nearest neighbors (kNN) operation.

Table IIIa confirms that PointGL outperforms the model variant, achieving higher accuracy while significantly reducing model complexity and improving efficiency. Specifically, while maintaining an equivalent parameter count, PointGL reduces FLOPs by an impressive factor of $11 \times$ and demonstrates approximately $2.7 \times$ faster training speed, coupled with an inference speed enhancement of approximately $2.5 \times$. These results effectively validate the critical importance of the global point embedding design within the PointGL framework.

To examine the effect of the number of residual MLP blocks on point embedding and its consequent impact on performance, we manipulated the quantity of these blocks within each learning stage, creating PointGL variants with 16, 24, and 32 layers by adjusting the number of blocks to 1, 2, and 3, respectively. As illustrated in Table IIIb, the addition of extra embedding layers does not consistently result in improved accuracy.



Fig. 3. Salient geometric characteristics attained during the initial learning stage of PointGL on the ModelNet40 dataset. Each point's color corresponds to its received vote count, wherein a more intense red hue signifies a greater vote count, while a deeper blue shade indicates a lower count of votes.

 TABLE IV

 Part segmentation results on the ShapeNetPart dataset.

Method	Inst. mIoU(%)	Cls. mIoU(%)	Infer Speed
PointNet [22]	83.7	80.4	-
DGCNN [11]	85.2	82.3	-
KPConv [25]	86.4	85.1	-
GDANet [45]	86.5	85.0	100.8
CurveNet [40]	86.6	-	60.0
PointNet++ [8]	85.1	81.9	276.3
PointMLP [14]	86.1	84.6	179.6
PointGL	85.6	83.8	284.6
PointGL-elite	85.0	83.0	309.0

Local Graph Pooling. The findings from Table IIIc reveal the results of integrating each constituent aspect of local graph pooling into a foundational architecture that employs direct max pooling after the kNN operation. Evidently, the creation of graphs to encapsulate point-to-point relationships emerges as a crucial factor within PointGL, leading to a significant improvement of the base architecture's overall accuracy by 4.0%. The inclusion of learnable parameters, specifically α and β , results in an additional enhancement of 0.8% in overall accuracy, resulting in a peak accuracy of 86.9%. This strongly underscores the validity of our design decisions.

Furthermore, we represent $\alpha \in \mathbb{R}^{D_s}$ and $\beta \in \mathbb{R}^{D_s}$ as D_s -dimensional learnable vectors, employed for channel-wise adjustment of responses in differential features. To assess the efficacy of our design, ablative experiments were conducted by substituting both α and β with learnable scalars. The experimental results indicate a noticeable performance degradation due to this modification, resulting in an overall accuracy reduction of 0.7%. These experiments affirm the effectiveness of the channel-wise feature adjustment design.

Next, we proceed to further validate the impact of the values of K, representing the number of neighboring points in the kNN algorithm. As presented in Table IIId, both excessively small (K = 12) and large (K = 36) values for K resulted in a



Fig. 4. Visualization of part segmentation outcomes on the ShapeNetPart dataset. In contrast to PointNet++, PointGL's predictions exhibit a more robust alignment with the ground truth.

 TABLE V

 Semantic segmentation results on the S3DIS Area-5 dataset.

Method	mIoU (%)	OA (%)	mAcc (%)	#Params	FLOPs	Infer Speed
PointNet [22]	41.1	-	49.0	3.6M	-	205.6
DGCNN [11]	47.9	83.6	-	1.3M	44.9G	10.6
ASSANet-L [37]	66.8	-	-	766.4M	86.0G	47.9
KPConv [25]	67.1	-	72.8	14.9M	-	-
RepSurf-U [41]	68.9	90.2	76.0	1.0M	3.4G	128.6
PT [13]	70.4	90.8	76.5	7.8M	3.1G	47.2
PointNet++ [8]	53.5	83.0	-	3.0M	6.5G	169.4
PointGL	65.6	88.6	71.9	3.5M	2.5 G	184.1

decrease in predictive accuracy. This occurs because small K values may lead to a diminished receptive field for features, while large K values may attenuate local detailed features. Optimal predictive accuracy was achieved with a suitable value for K = 24. As a result, we adopted K = 24 in our experiments.

We have successfully implemented the local graph pooling as a versatile drop-in block, distinguished by its minimal parameter requirement and enhanced efficiency. To assess its adaptability in point-based models like PointNet++ and PointMLP, we replaced the native max pooling with our local graph pooling, excluding the sampling and kNN operations. The results shown in Table IIIe demonstrate that the straightforward incorporation of local graph pooling consistently enhances performance across a range of models. This underscores the universal nature of our pooling approach, highlighting its seamless applicability within existing point-based models to improve feature aggregation, all while incurring minimal additional parameters and computational overhead.

Visualization of Features. To deepen our understanding of the behavior of our hierarchical network, Fig. 3 offers a visual representation of the focal areas to which the network directs its attention during its initial learning stage. This was achieved by gathering the indices generated by the max pooling process within each local graph and subsequently consolidating these indices across all local graphs, resulting in votes for each input

Method	Car (IoU=0.7)			Р	edestrian (IoU=0.	5)	Cyclist (IoU=0.5)			
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard	
Second* [46]	88.83	78.60	77.33	57.88	53.26	49.00	81.05	67.62	63.06	
PointPillar* [47]	87.05	77.29	75.65	56.49	50.83	46.84	79.56	63.35	59.53	
Part- A^2 -free* [48]	89.07	78.64	78.10	68.02	63.13	58.32	86.70	72.33	69.59	
PointRCNN* [49]	88.57	78.53	77.75	60.38	53.44	49.36	87.89	73.29	67.65	
+ LGP	88.72	78.55 \0.02	77.66	61.60	54.38 ↑0.94	49.86	87.54	73.63 ↑0.34	71.31	
PV-RCNN* [50]	89.58	83.19	78.86	63.72	57.35	53.40	84.93	71.67	68.29	
+ LGP	89.43	83.49 \(0.30 \)	78.86	65.87	59.35 ↑2.00	54.56	86.36	72.67 1.00	69.09	

 TABLE VI

 PERFORMANCE ON KITTI Validation SET. * DENOTES RESULTS FROM OPENPCDET.



Fig. 5. Illustrative outcomes of 3D object detection on the KITTI dataset. Ground-truth and predicted objects are distinguished by red and green boxes, respectively. Integration of local graph pooling effectively recovers diminutive objects that were overlooked by the PV-RCNN baseline in intricate distant scenarios.

point. Points that contribute more significantly to the local representation, embodying recognized patterns, accumulate increased votes. The highlighted structural elements encompassing planes, lines, corners, and similar features affirm the network's proficiency in discerning and encapsulating crucial local geometric attributes.

D. Object Part and Scene Segmentation

The versatility of our PointGL framework allows for its extension to a variety of tasks. To assess its performance in the context of 3D shape part segmentation, we conducted experiments on the ShapeNetPart dataset [51]. This dataset comprises a collection of 16, 881 shapes distributed across 16 distinct classes, each annotated with 50 parts. We randomly sampled 2, 048 points as inputs [8] and trained the model for 350 epochs using the Adam [52] optimizer with a batch size of 32. The optimizer is configured with betas=(0.9, 0.999) and learning rate of 0.003. The stepLR scheduler is also employed to decrease the learning rate with step size of 40 and gamma of 0.5.

As demonstrated in Table IV, our PointGL achieved remarkable performance, boasting an instance average mean Intersection over Union (mIoU) of 85.6%, while also maintaining notable inference speed. Furthermore, our approach outperformed the PointNet++ baseline in the majority of categories. Fig. 4 visually reinforces how PointGL's predictions closely align with the ground truth.

Moreover, our PointGL approach underwent evaluation on the S3DIS Area-5 dataset [53], which pertains to 3D semantic segmentation. In our experimental setup, we train the PointGL model with a batch size of 32 over 100 epochs, employing the AdamW optimizer. The optimizer is specifically configured with a learning rate of 0.01 and a weight decay rate of 1e-4. To manage the learning rate schedule, we utilize the CosineLRScheduler scheme, which reduces the learning rate progressively until it reaches a minimum value of 1e-5. As presented in Table V, PointGL achieved remarkably competitive performance, achieving an mIoU of 65.6%, surpassing established benchmarks like PointNet++ (53.5%), while simultaneously performing on par with contemporary methods like RepSurf-U. Significantly, these achievements were reached with a lower FLOP count and maintained high-speed inference. These results underscore the adaptability of our PointGL approach and its potential for utilization across a spectrum of

TABLE VII CLASSIFICATION PERFORMANCE UNDER REAL-WORLD CORRUPTIONS ON THE MODELNET40-C DATASET.

Method	OA(%)	mCE↓	Sca	Jit	D-G	D-L	A-G	A-L	Rot
DGCNN [11]	92.6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PointNet [22]	90.7	1.42	1.27	0.64	0.50	1.07	2.98	1.59	1.90
PointNet++ [8]	93.0	1.07	0.87	1.18	0.64	1.80	0.61	0.99	1.41
RSCNN [35]	92.3	1.13	1.07	1.17	0.81	1.52	0.71	1.15	1.48
GDANet [45]	93.4	0.89	0.83	0.84	0.79	0.89	0.87	1.04	0.98
SimpleView [55]	93.9	1.05	0.87	0.72	1.24	1.36	0.98	0.84	1.32
PAConv [10]	93.6	1.10	0.90	1.47	1.00	1.01	1.09	1.30	0.97
CurveNet [40]	93.8	0.93	0.87	0.73	0.71	1.02	1.35	1.00	0.81
PCT [12]	93.0	0.93	0.87	0.87	0.53	1.00	0.78	1.39	1.04
RPC [34]	93.0	0.86	0.84	0.89	0.49	0.80	0.93	1.01	1.08
PointGL	93.4	0.77	1.01	1.15	0.60	1.19	0.24	0.26	0.94

downstream tasks.

E. Object Detection

Data and Setup. Our proposed *Local Graph Pooling* operation seamlessly integrates into point cloud backbones, facilitating the extraction of intricate geometric features that can profoundly benefit downstream tasks, such as object detection. To demonstrate the effectiveness of our approach, we replaced the native set abstraction layer with our proposed module in prominent detection frameworks, specifically PV-RCNN [50] and PointRCNN [49]. We then evaluated the resulting object detectors on the widely used KITTI dataset [54]. The dataset comprises a training set of 7, 481 samples, conventionally split into 3, 712 and 3, 769 samples for training and validation, respectively. Our detector was trained for 80 epochs using the Adam optimizer, initialized with a learning rate of 0.01. The weight decay of the optimizer is set to 0.01. The evaluation metric employed was the per-class Average Precision (AP).

Results. The outcomes of integrating our innovative *Local Graph Pooling* operation into the PV-RCNN [50] baseline are detailed in Table VI. This integration resulted in significant improvements of 0.3, 2.0, and 1.0 in AP for the Car, Pedestrian, and Cyclist categories, respectively. These results strongly emphasize the effectiveness of our approach in enhancing the acquired feature representation with intricate geometric details, which substantially contribute to the successful detection of objects of various scales. Moreover, the benefits of our proposed methodology extend beyond PV-RCNN, as it also yields improvements in other state-of-the-art detectors like PointRCNN, highlighting its versatile applicability across diverse detection frameworks.

Visualization. Qualitative results obtained through the application of our approach are vividly depicted in Fig. 5. This visualization distinctly illustrates instances where the PV-RCNN baseline faces challenges in detecting small objects, especially in scenarios with sparse point density. In contrast, the incorporation of our proposed *Local Graph Pooling* operation leads to a significant improvement in detection performance in such scenarios. This empirical observation strongly emphasizes the effectiveness of our method in capturing crucial geometric details necessary for achieving accurate long-range object

TABLE VIII SEGMENTATION PERFORMANCE UNDER REAL-WORLD CORRUPTIONS ON THE SHAPENET-C DATASET.

Method	mCE↓	Scale	Jitter	Drop-G	Drop-L	Add-G	Add-L	Rotate
DGCNN [11]	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PointNet [22]	1.18	1.08	1.05	0.98	1.13	1.39	1.17	1.44
PointNet++ [8]	1.11	0.95	1.08	0.86	1.98	0.89	1.08	0.95
PAConv [10]	0.93	0.93	1.07	0.93	0.93	0.74	0.95	0.95
GDANet [45]	0.92	0.92	1.01	0.94	0.95	0.71	0.96	0.97
PT [13]	1.05	1.08	1.07	1.03	1.08	1.11	1.07	0.91
Point-MLP [14]	0.98	0.97	1.13	0.89	0.99	0.93	1.06	0.88
OcCo-DGCNN [57]	0.98	0.96	1.07	0.96	1.02	0.94	1.00	0.89
Point-BERT [58]	1.03	0.94	1.10	0.87	0.93	1.17	1.20	1.03
Point-MAE [59]	0.93	0.91	1.04	0.85	0.88	0.78	1.03	1.00
PointGL	0.82	0.99	1.10	0.94	1.05	0.42	0.43	0.84

detection, even in situations characterized by limited dense point data.

F. Robustness Analysis

Real-world applications dealing with point cloud data often encounter challenges arising from sensor inaccuracies and complex scene structures, leading to data corruptions. Thus, the ability to handle point cloud corruptions effectively becomes crucial for practical applications. To assess the resilience and versatility of our proposed PointGL approach, we conducted a series of experiments on the ModelNet-C [34] and ShapeNet-C [56] datasets. Our evaluation covers tasks such as point cloud classification and part segmentation, conducted under various corruption scenarios.

Results on ModelNet-C. We assess the robustness of our proposed PointGL approach using the ModelNet-C [34] point cloud classification corruption test suite. This suite encompasses seven distinct types of corruptions, each spanning five severity levels. This comprehensive assessment framework provides a rigorous evaluation of the model's inherent robustness. Our model is trained on the clean ModelNet40 [44] dataset and subsequently evaluated on the ModelNet-C [34] test suite. Training utilizes the SGD optimizer with a batch size of 32 and a learning rate of 0.1, conducted over 300 epochs.

Table VII presents a comprehensive summary of the evaluation results for point cloud classification models applied to the ModelNet-C dataset. The results highlight a significant observation - many state-of-the-art methods, known for their high Overall Accuracy (OA) on the clean ModelNet40 dataset, exhibit notable vulnerability when subjected to corruptions. In contrast, our proposed PointGL excels by achieving the lowest mean Corruption Error (mCE) of 0.77 across all corruption scenarios. This accomplishment unquestionably demonstrates the robustness of our model. These findings underscore the substantial potential of PointGL in practical real-world scenarios.

Results on ShapePart-C. The ShapeNet-C [56] benchmark is specifically designed to systematically assess the robustness of point cloud segmentation models across a range of corruptions. This benchmark encompasses seven distinct corruption categories, each calibrated at five different severity levels. To rigorously evaluate the robustness of our proposed PointGL, we conducted an extensive set of experiments on the ShapeNet-C [56] dataset, focusing on the task of part segmentation. Throughout the model training procedure, we employed the AdamW optimizer for a duration of 300 epochs, with batch size of 16. The optimizer's configuration comprised a learning rate of 0.0002 and a weight decay coefficient of 0.05. Furthermore, we applied the CosineLRScheduler scheme to modulate the learning rate, gradually reducing it to a minimum value of 1e-6. The warm-up epoch period is set to 10.

The performance of our proposed PointGL on the ShapeNet-C benchmark is highlighted in Table VIII. The achieved classwise mIoU score of 0.82 significantly outperforms the stateof-the-art PointMAE method by a substantial margin of 0.11. Notably, PointMAE is a masked autoencoder (MAE) model trained on a substantial volume of unlabeled point cloud data. This noteworthy achievement underscores that PointGL effectively harnesses vital local information while preserving the integrity of semantic information. The demonstrated performance not only speaks to the robustness of our approach but also underscores its broad applicability across different tasks.

V. CONCLUSION

In this paper, we introduce PointGL, an architecture that combines simplicity and potency to employ a novel compact paradigm for efficient point cloud analysis. Our approach initiates by generating feature embeddings for individual points through residual MLPs. Subsequently, we introduce an innovative technique called local graph pooling, aimed at capturing regional features while minimizing extra learnable parameters and computational overhead. Our experiments on diverse benchmarks consistently demonstrate PointGL's superior performance compared to previous state-of-the-art models, achieving this with significantly reduced model complexity and heightened efficiency. We anticipate that our PointGL architecture will serve as an inspiration for the community to reevaluate efficient network design strategies tailored to point clouds.

REFERENCES

- S. Deng, Q. Dong, B. Liu, and Z. Hu, "Superpoint-guided semisupervised semantic segmentation of 3d point clouds," in *ICRA*, 2022.
- [2] Y. Zhao, X. Zhang, and X. Huang, "A divide-and-merge point cloud clustering algorithm for lidar panoptic segmentation," in *ICRA*, 2022.
- [3] X. Chen, H. Zhao, G. Zhou, and Y.-Q. Zhang, "Pq-transformer: Jointly parsing 3d objects and layouts from point clouds," *IEEE RAL*, 2022.
- [4] Q. Yang, H. Chen, Z. Ma, Y. Xu, R. Tang, and J. Sun, "Predicting the perceptual quality of point cloud: A 3d-to-2d projection-based exploration," *IEEE TMM*, 2020.
- [5] S. Qiu, S. Anwar, and N. Barnes, "Geometric back-projection network for point cloud classification," *IEEE TMM*, 2021.
- [6] X.-F. Han, Y.-F. Jin, H.-X. Cheng, and G.-Q. Xiao, "Dual transformer for point cloud analysis," *IEEE TMM*, 2022.
- [7] M. Shabbir, A. Shabbir, C. Iwendi, A. R. Javed, M. Rizwan, N. Herencsar, and J. C.-W. Lin, "Enhancing security of health information using modular encryption standard in mobile cloud computing," *IEEE Access*, vol. 9, pp. 8820–8834, 2021.
- [8] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *NeurIPS*, 2017.

- [9] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," in *CVPR*, 2019.
- [10] M. Xu, R. Ding, H. Zhao, and X. Qi, "Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds," in *CVPR*, 2021.
- [11] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," ACM TOG, 2019.
- [12] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "Pct: Point cloud transformer," *Computational Visual Media*, 2021.
- [13] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *ICCV*, 2021.
- [14] X. Ma, C. Qin, H. You, H. Ran, and Y. Fu, "Rethinking network design and local geometry in point cloud: A simple residual mlp framework," in *ICLR*, 2022.
- [15] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *IROS*, 2015.
- [16] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *CVPR*, 2015.
- [17] H. Guo, J. Wang, Y. Gao, J. Li, and H. Lu, "Multi-view 3d object retrieval with deep embedding network," *TIP*, 2016.
- [18] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," in *CVPR*, 2016.
- [19] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *ICCV*, 2015.
- [20] A. Saha, O. Mendez, C. Russell, and R. Bowden, "Translating images into maps," in *ICRA*, 2022.
- [21] L. Wiesmann, R. Marcuzzi, C. Stachniss, and J. Behley, "Retriever: Point cloud retrieval in compressed 3d maps," in *ICRA*, 2022.
- [22] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in CVPR, 2017.
- [23] Z. Liu, H. Hu, Y. Cao, Z. Zhang, and X. Tong, "A closer look at local aggregation operators in point cloud analysis," in ECCV, 2020.
- [24] M. Meraz, M. A. Ansari, M. Javed, and P. Chakraborty, "Dc-gnn: drop channel graph neural network for object classification and part segmentation in the point cloud," *International Journal of Multimedia Information Retrieval*, vol. 11, no. 2, pp. 123–133, 2022.
- [25] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *ICCV*, 2019.
- [26] G. Qian, Y. Li, H. Peng, J. Mai, H. Hammoud, M. Elhoseiny, and B. Ghanem, "Pointnext: Revisiting pointnet++ with improved training and scaling strategies," *NeurIPs*, 2022.
- [27] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "Spidercnn: Deep learning on point sets with parameterized convolutional filters," in ECCV, 2018.
- [28] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," *NeurIPS*, 2018.
- [29] Z.-H. Lin, S. Y. Huang, and Y.-C. F. Wang, "Learning of 3d graph convolution networks for point cloud analysis," *IEEE TPAMI*, 2021.
- [30] H. Ran, W. Zhuo, J. Liu, and L. Lu, "Learning inner-group relations on point clouds," in *ICCV*, 2021.
- [31] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," arXiv preprint arXiv:1903.12261, 2019.
- [32] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do imagenet classifiers generalize to imagenet?" in *ICML*, 2019.
- [33] A. Barbu, D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum, and B. Katz, "Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models," *NeurIPs*, 2019.
- [34] J. Ren, L. Pan, and Z. Liu, "Benchmarking and analyzing point cloud classification under corruptions," arXiv preprint arXiv:2202.03377, 2022.
- [35] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in CVPR, 2019.
- [36] J. Wang, J. Li, L. Ding, Y. Wang, and T. Xu, "Papooling: Graph-based position adaptive aggregation of local geometry in point clouds," *arXiv* preprint arXiv:2111.14067, 2021.
- [37] G. Qian, H. Hammoud, G. Li, A. Thabet, and B. Ghanem, "Assanet: An anisotropic separable set abstraction for efficient point cloud representation learning," in *NeurIPS*, 2021.
- [38] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui, "Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling," in CVPR, 2020.

- [39] A. Hamdi, S. Giancola, and B. Ghanem, "Mvtn: Multi-view transformation network for 3d shape recognition," in *ICCV*, 2021.
- [40] T. Xiang, C. Zhang, Y. Song, J. Yu, and W. Cai, "Walk in the cloud: Learning curves for point clouds shape analysis," in *ICCV*, 2021.
- [41] H. Ran, J. Liu, and C. Wang, "Surface representation for point clouds," in CVPR, 2022.
- [42] M. A. Uy, Q.-H. Pham, B.-S. Hua, T. Nguyen, and S.-K. Yeung, "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data," in *ICCV*, 2019.
- [43] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," arXiv preprint arXiv:1608.03983, 2016.
- [44] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *CVPR*, 2015.
- [45] M. Xu, J. Zhang, Z. Zhou, M. Xu, X. Qi, and Y. Qiao, "Learning geometry-disentangled representation for complementary understanding of 3d object point cloud," in AAAI, 2021.
- [46] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, 2018.
- [47] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *CVPR*, 2019.
- [48] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, "From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network," *IEEE TPAMI*, 2020.
- [49] S. Shi, X. Wang, and H. Li, "Pointrenn: 3d object proposal generation and detection from point cloud," in CVPR, 2019.

- [50] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection," in *CVPR*, 2020.
- [51] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas, "A scalable active framework for region annotation in 3d shape collections," ACM TOG, 2016.
- [52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
- [53] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3d semantic parsing of large-scale indoor spaces," in *CVPR*, 2016.
- [54] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, 2013.
- [55] A. Goyal, H. Law, B. Liu, A. Newell, and J. Deng, "Revisiting point cloud shape classification with a simple and effective baseline," in *ICML*, 2021.
- [56] J. Ren, L. Kong, L. Pan, and Z. Liu, "Pointcloud-c: Benchmarking and analyzing point cloud perception robustness under corruptions," *preprint*, 2022.
- [57] H. Wang, Q. Liu, X. Yue, J. Lasenby, and M. J. Kusner, "Unsupervised point cloud pre-training via occlusion completion," in *ICCV*, 2021.
- [58] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu, "Point-bert: Pretraining 3d point cloud transformers with masked point modeling," in *CVPR*, 2022.
- [59] Y. Pang, W. Wang, F. E. Tay, W. Liu, Y. Tian, and L. Yuan, "Masked autoencoders for point cloud self-supervised learning," in ECCV, 2022.