# PointODE: Lightweight Point Cloud Learning with Neural Ordinary Differential Equations on Edge

Keisuke Sugiura*, Mizuki Yasuda†, and Hiroki Matsutani†
* Institute of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki, 305-8573, Japan
† Department of Information and Computer Science, Keio University, Yokohama, Kanagawa, 223-8522, Japan

***Abstract*—Embedded devices are often used as a platform to perform real-world point cloud tasks, but recent deep learning-based methods may not fit on such devices. In this paper, we aim to fill this gap by introducing PointODE, a parameter-efficient ResNet-like architecture for point cloud feature extraction based on a stack of MLP blocks with residual connections. We leverage Neural ODE (Ordinary Differential Equation), a continuous-depth version of ResNet originally developed for modeling the dynamics of continuous-time systems, to compress PointODE by reusing the same parameters across multiple MLP blocks. We introduce PointODE-Elite as a lightweight version with 0.58M trainable parameters and design its dedicated FPGA accelerator, which consists of a four-stage pipeline to parallelize feature extraction. Compared to the ARM Cortex-A53 CPU, the accelerator implemented on a Xilinx ZCU104 board speeds up the feature extraction by 4.9x, leading to 3.7x faster inference and 3.5x better energy-efficiency. Despite the simple architecture, PointODE-Elite shows competitive accuracy to state-of-the-art models on both synthetic and real-world classification datasets.**

***Index Terms*—FPGA, Point Cloud, PointMLP, Residual Networks, Neural Ordinary Differential Equations**

## I. Introduction

Thanks to the increasing availability of low-cost 3D scanners (e.g., LiDARs and depth cameras), point cloud serves as a basis for various mobile-edge applications including 3D reconstruction [1], mapping [2], and object tracking [3]. Since PointNet [4] and PointNet++ [5], deep learning-based point cloud analysis has been the subject of extensive research and gained tremendous success. PointNet is the first successful architecture for learning a global representation of point clouds via MLP-based point-wise feature extraction and global pooling. PointNet++ adopts a hierarchical PointNet architecture to capture local geometric structures at varying scales. The follow-up methods employ various techniques such as custom convolution operators [6], graph convolutions [7], and transformers [8] to enhance the feature representation, at the cost of increasing model complexity.

PointMLP [9] takes the opposite direction and avoids the use of complicated feature extractors. It is composed of a stack of MLP blocks with residual connections, but achieves competitive accuracy to state-of-the-art models on tasks such as classification and part segmentation. In addition, PointMLP can efficiently leverage the computing power of FPGAs, as it mainly involves highly-parallelizable matrix operations within MLPs. PointMLP is basically a ResNet-like architecture with MLPs instead of 2D convolutions.

Interestingly, ResNet can be interpreted as a discrete approximation of an ODE. Based on this close relationship, ResNet is extended to a continuous-depth version dubbed as Neural ODE [10], to represent the hidden dynamics of continuous-time systems. It is widely adopted in a variety of tasks, e.g., climate forecasting [11], physics simulation [12], and video generation [13]. Neural ODE is inherently more parameter-efficient than ResNet, as it is equivalent to a ResNet with an arbitrary number of residual blocks sharing the same parameters. From this perspective, Neural ODE can be viewed as a model compression technique for ResNet-like models. Unlike recent complicated feature extractors, Neural ODE can be easily applied to PointMLP with minimal changes thanks to its simple architecture.

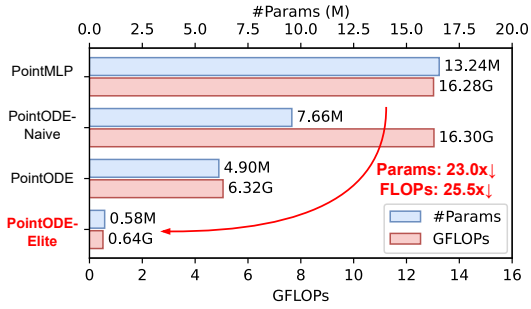


Fig. 1: Number of parameters and FLOPs of the proposed PointODE-Elite and the baseline PointMLP.

In this paper, we introduce a new DNN architecture for point cloud analysis, **PointODE**, combining the advantages of both PointMLP and Neural ODE. PointODE is a simple residual MLP-based network but integrates Neural ODE to effectively reduce the number of parameters by replacing a sequence of residual blocks with a single reusable block (Fig. 1). By adjusting the network size, we propose a lightweight version named **PointODE-Elite** with 0.58M learnable parameters, significantly improving the accuracy-computation trade-off.

We then design a dedicated FPGA accelerator for PointODE-Elite, which incorporates a set of MLP blocks and the Euler numerical integration method for solving ODEs. PointODE omits the geometric affine transformation in PointMLP and instead performs a point-wise normalization to avoid the necessity of computing global statistics of input feature points, leading to the improved parallelism and accuracy. The design consists of a four-stage feature extraction pipeline to exploit the point-level parallelism. Thanks to the lower parameter size and reuse of blocks, parameters and intermediate activations of the entire model fit on-chip. The proposed accelerator is implemented on a Xilinx ZCU104 board and evaluated on classification datasets. It runs 4.9x faster than

the ARM Cortex-A53 processor, while achieving a comparable accuracy to state-of-the-art networks. The contributions of this paper are summarized as follows:

- We propose PointODE and its lightweight counterpart PointODE-Elite as a simple yet highly-accurate DNN architecture for point clouds.
- To the best of our knowledge, this paper is the first to explore an FPGA design for the ODE-based point cloud analysis.

## II. Related Work

**DNNs for Point Clouds**: PointNet [4] is a pioneering model that directly consumes point clouds. It extracts point-wise features using a shared MLP and aggregates them into a global representation via a symmetric pooling function. PointNet++ [5] is an extension that aims to capture fine geometric context through repeated sampling and grouping of feature points. They have catalyzed the development of more sophisticated networks to learn better feature representations at the cost of increased complexity.

One approach is to design custom convolution kernels for point clouds [6], [14]–[16], while several works [7], [17], [18] employ GCNs (Graph Convolutional Networks) to process KNN graphs built from point clouds. Transformer-based methods [19], [20] employ a self-attention mechanism to capture the relationship between all points, while it incurs a quadratic computational cost in terms of input size. Another approach is to represent point clouds as curves [21], polygon meshes [22], or umbrella surfaces [23]. Aside from these approaches, PointNeXt [24] revisit PointNet++ and proposes improved training strategies to boost its performance without changing the network structure. PointMLP [9] employs MLPs with residual connections and learns a hierarchical feature representation like PointNet++. It achieves competitive accuracy and faster inference speed compared to the existing complex feature extractors, demonstrating the effectiveness of a simple MLP-based architecture.

**ODE-based DNNs**: Neural ODE [10] is initially developed as a method to model the hidden dynamics (i.e., derivatives) of a continuous-time system using a neural network. Various follow-up works have emerged to e.g., enhance the expressive power [25], obtain higher-order derivatives [26], and avoid incorrect gradients and numerical instability [27], [28]. ExNODE [29] is a seminal work that extends Neural ODE to point clouds. CaSPR [30] uses Neural ODE to learn the representations of dynamically moving point clouds. Exploiting the connection between ResNets and Neural ODEs, the recent work [31], [32] utilizes Neural ODE as a technique to compress ResNet-like image classification models without sacrificing accuracy. Taking a similar approach, this work integrates Neural ODE into PointMLP and builds a lightweight network to perform point cloud analysis on embedded devices.

**Neural ODEs on FPGAs**: Only a few works [31]–[34] explore the FPGA design of Neural ODEs. The authors of [31] combine Neural ODE and separable convolution to develop a lightweight network with only 0.6M parameters, which is implemented on a Xilinx ZCU104 board. In [32], an FPGA accelerator is proposed for a CNN-Transformer hybrid model combining Neural ODE and ViT (Vision Transformer) [35]. In [33], the third-order Runge-Kutta method is used as an ODE solver instead of the first-order Euler method, while this requires 3x more network forward passes per iteration. A two-stage structured pruning method along with a history-based step size search is devised to mitigate this problem. While these accelerators achieve favourable performance, they are only evaluated on small-scale image datasets (e.g., CIFAR-10) and not designed for 3D point clouds.

## III. Background

### A. Neural ODE

ResNet employs residual connections in its building blocks (**ResBlock**s) to stabilize the training of deep networks. ResBlocks can be expressed as the following recursive formula:

$$\mathbf{h}_t = \mathbf{h}_{t-1} + f(\mathbf{h}_{t-1}, \boldsymbol{\theta}_t), \tag{1}$$

where $t = 1, \ldots$ denotes a block index, $\mathbf{h}_{t-1}$ an input, $f$ a stack of layers, and $\boldsymbol{\theta}_t$ a set of trainable parameters in a block. The addition represents a residual connection. Eq. 1 can be viewed as a forward Euler discretization of the ODE $\mathrm{d}\mathbf{h}(t) / \mathrm{d}t = f(\mathbf{h}(t), t, \boldsymbol{\theta})$, where $t$ denotes a continuous time and $f(\cdot, \boldsymbol{\theta})$ represents a time-derivative (i.e., dynamics) of a hidden state $\mathbf{h}(t)$. Neural ODE [10] formulates the forward propagation of ResBlocks as a solution of this ODE. Given an input $\mathbf{h}(t_a)$, an output $\mathbf{h}(t_b)$ is obtained using an arbitrary ODE solver (i.e., by integrating $f$ over $[t_a, t_b]$). For instance, the Euler method can be used:

$$\mathbf{h}(t_j) = \mathbf{h}(t_{j-1}) + hf(\mathbf{h}(t_{j-1}), t_{j-1}, \boldsymbol{\theta}), \tag{2}$$

where $j = 0, \ldots, C$ denotes a discrete time step ($t_j = t_a + jh$, $h = (t_b - t_a)/C$) and $C$ is the number of iterations. Eq. 2 represents an ODE-based building block (**ODEBlock**) consisting of layers $f$, parameters $\boldsymbol{\theta}$, and a residual connection. As depicted in Fig. 2, ODEBlock uses the same network $f$ during $C$ forward passes, which is $C$ times more parameter-efficient than using $C$ separate ResBlocks.

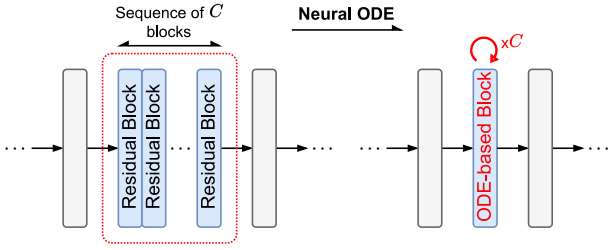


Fig. 2: Replacing the forward pass of $C$ ResBlocks with $C$ forward iterations of the single ODE-based building block.

### B. PointMLP

PointMLP [9] is a pure MLP-based network with residual connections for point cloud feature extraction. The architecture is presented in Fig. 3. It directly takes 3D point coordinates $\mathcal{P} = \{\mathbf{p}_i\}_{i=1}^{N}$ as input and extracts high-dimensional features for a subset of points, which are fed to the network for a specific task (e.g., classification and segmentation). PointMLP consists of an embedding block to extract $F_0$-dim local features $\{\mathbf{f}_i^0\}_{i=1}^{N}$ for each point, which is followed by a stack of four stages to hierarchically aggregate these local features.

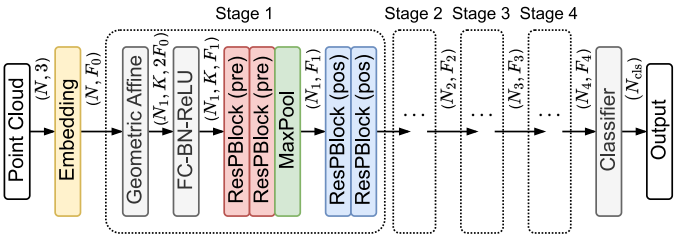


Fig. 3: Architecture of PointMLP.

Each stage $s \in [1, 4]$ produces $F_s$-dim features $\mathcal{F}^s = \{\mathbf{f}_i^s\}$ for $N_s$ sampled points $\mathcal{P}^s = \{\mathbf{p}_i^s\}$. PointMLP first samples $N_s$ points (i.e., group centroids) from $\mathcal{P}^{s-1}$ and finds their $K$-nearest neighbors (NNs). For each group centroid $\mathbf{p}_j^{s-1} \in \mathcal{P}^{s-1}$, the geometric affine module applies an affine transformation to the features $\{\mathbf{f}_{j,k}^{s-1}\}$ of its neighbors $\{\mathbf{p}_{j,k}^{s-1}\}$. The transformed features $\{\hat{\mathbf{f}}_{j,k}^{s-1}\}$ of grouped points are then concatenated with that of the centroid $\mathbf{f}_j^{s-1} \in \mathcal{F}^{s-1}$, producing a tensor of size $(N_s, K, 2F_{s-1})$. As shown in Fig. 3 (**FC-BN-ReLU** and **ResPBlock (pre)**), an MLP block and the first two residual point blocks (**ResPBlock**s) transform these $2F_{s-1}$-dim features into $F_s$-dim features and produce an output of size $(N_s, K, F_s)$. The max-pooling aggregates features of $K$ local neighbors into a single $F_s$-dim feature that capture the geometry of the whole group. The subsequent two ResPBlocks (**ResPBlock (pos)** in Fig. 3) extract deeply aggregated features and produces an output $\mathcal{F}^s$ of size $(N_s, F_s)$.

PointMLP operates on raw point clouds instead of 3D grids or 2D images and hence does not require costly preprocessing (e.g., voxelization and multi-view rendering). Besides, it does not rely on sophisticated feature extractors (e.g., custom convolution operators [6], [14], [15], [36]). ResPBlock (Fig. 4, left) has a simple architecture consisting of FC layers along with BN (Batch Normalization) and ReLU. It offers high parallelism as the feature extraction is performed independently for each point. PointMLP is thus amenable to FPGA acceleration thanks to its simple architecture and high parallelism.

## IV. PointODE-Elite

This section describes the architecture of PointODE-Elite. For parameter-efficiency and parallelism, it introduces the three improvements: (1) ODE-based residual point block, (2) residual block reordering, and (3) point-wise normalization.

### A. ODEPBlock: ODE-based Residual Point Block

Each stage in PointMLP has two sets of two ResPBlocks for extracting local and aggregated features (Fig. 3). Using Neural ODE, a single forward pass of two consecutive ResPBlocks can be replaced by two forward iterations of a single ODE-based residual point block (**ODEPBlock**). Since ResPBlocks dominate the model size of PointMLP (i.e., account for 88.9% of the total parameters), this leads to a 1.73x parameter reduction (13.24M to 7.66M). We refer to this model as **PointODE-Naive**. Note the number of forward iterations $C$ allows to balance the trade-off between accuracy and inference time (Fig. 11). As depicted in Fig. 4, ODEPBlock is similar to ResPBlock but has two additional layers (**ConcatT**) to fuse the time information (i.e., a continuous time variable $t$) into the input features. Formally, ODEPBlock in stage $s$ takes $F_s$-dim features for $N_s$ points and produces an output of the same size. The first ConcatT layer concatenates a time variable $t$ to the input, creating a set of $(F_s + 1)$-dim features, which is passed to an FC-BN-ReLU block to extract $F_s'$-dim features. Followed by another ConcatT layer, an FC-BN block transforms $(F_s' + 1)$-dim features into $F_s$-dim ones, which are then added to the original input via a residual connection.

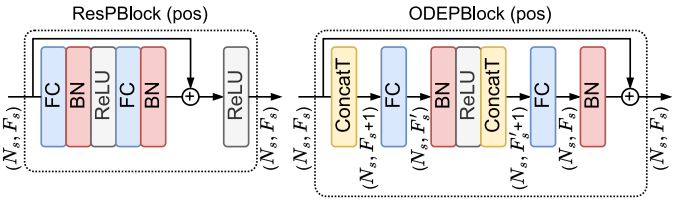


Fig. 4: ResPBlock and ODEPBlock.

### B. Residual Block Reordering

To improve the effectiveness of using ODE blocks, we propose to first reorder the blocks in each stage of PointMLP. As shown in Fig. 5, the first two ResPBlocks are moved next to the max-pooling layer, such that four ResPBlocks (now stacked in sequence) can be replaced by four forward iterations of the same ODEPBlock. This model has 1.56x fewer parameters than PointODE-Naive (7.66M to 4.90M), as each stage has only one ODEPBlock instead of two[1]. The resulting model architecture is shown in Fig. 6. An MLP block (FC-BN-ReLU) extracts $F_s$-dim local features for each of $K$ neighboring points of a group centroid, producing an output of size $(N_s, K, F_s)$, while an ODEPBlock learns an aggregated feature for each group. Following PointMLP, $F_s$ and $N_s$ are set to $2F_{s-1}$ and $N_{s-1}/2$, respectively, such that each stage extracts higher-level features for fewer representative points.

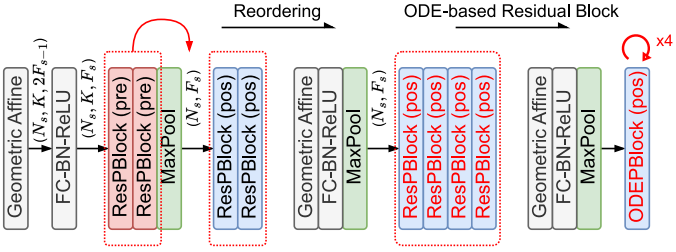


Fig. 5: Residual block reordering (left: PointMLP, center: reordered layers, right: PointODE(-Elite)).

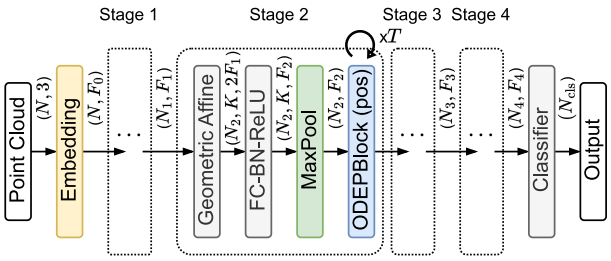


Fig. 6: Architecture of PointODE and PointODE-Elite.

[1]The first stage keeps two separate ODEPBlocks for accuracy.

### C. Point-wise Normalization

In each stage $s$ of PointMLP, the geometric affine module first transforms features $\{\mathbf{f}_{j,k}^{s-1}\}$ of the $K$ neighborhood points $\{\mathbf{p}_{j,k}^{s-1}\}$ in a group as follows:

$$\hat{\mathbf{f}}_{j,k}^{s-1} = \boldsymbol{\alpha} \odot \frac{\Delta\mathbf{f}_{j,k}^{s-1}}{\sigma + \varepsilon} + \boldsymbol{\beta}, \ \Delta\mathbf{f}_{j,k}^{s-1} = \mathbf{f}_{j,k}^{s-1} - \mathbf{f}_{j}^{s-1}, \quad (3)$$

where $\mathbf{f}_j^{s-1} \in \mathcal{F}^{s-1}$ is an $F_{s-1}$-dim feature associated with the group centroid $\mathbf{p}_j^{s-1} \in \mathcal{P}^{s-1}$. The two $F_{s-1}$-dim vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are trainable scale and offset parameters, respectively, and $\varepsilon$ = 1e-5 is a small constant to avoid zero division. The scalar $\sigma$ is defined as:

$$\boldsymbol{\mu} = \frac{\sum_{j,k} \Delta\mathbf{f}_{j,k}^{s-1}}{N_{s-1}K}, \ \sigma = \sqrt{\frac{\sum_{j,k} \left\|\Delta\mathbf{f}_{j,k}^{s-1} - \boldsymbol{\mu}\right\|_2^2}{N_{s-1}K}}. \quad (4)$$

Since $\boldsymbol{\mu}$ and $\sigma$ are the global mean and standard deviation of the feature residual $\Delta\mathbf{f}_{j,k}^{s-1}$ across all $N_{s-1}$ points, the affine transform is not fully independent for each point. Each stage cannot proceed to the normalization (Eq. 3) and subsequent blocks until $\sigma$ is computed. Besides, the accumulation of $N_{s-1}K$ features requires a wider fixed-point format to avoid overflow. Considering these, we propose to compute $\mu$ and $\sigma$ separately for each point in a group (similar to layer normalization [37]) and transform the features as follows:

$$\tilde{\mathbf{f}}_{j,k}^{s-1} = \boldsymbol{\alpha} \odot \frac{\Delta\mathbf{f}_{j,k}^{s-1}}{\tilde{\sigma}_{j,k} + \varepsilon} + \boldsymbol{\beta} \quad (5)$$

$$\tilde{\mu}_{j,k} = \frac{\mathbf{1}^\top \Delta\mathbf{f}_{j,k}^{s-1}}{F_{s-1}}, \ \tilde{\sigma}_{j,k} = \sqrt{\frac{\left\|\Delta\mathbf{f}_{j,k}^{s-1} - \tilde{\mu}_{j,k}\mathbf{1}\right\|_2^2}{F_{s-1}}}. \quad (6)$$

where $\mathbf{1}$ is an $F_{s-1}$-dim vector of ones. In this case, $\tilde{\mu}$ and $\tilde{\sigma}$ represent the mean and standard deviation of the elements in $\Delta\mathbf{f}_{j,k}^{s-1}$. The affine transform (Eq. 5) can be performed independently for each point, which allows for a pipelined execution of blocks in a stage (Sec. V). In addition, such normalization can better handle irregular point clouds with a varying point distribution by adjusting the scaling factor $\tilde{\sigma}_{j,k}$ for each point $\mathbf{p}_{j,k}$ in a group. We experimentally validate the accuracy improvements with point-wise normalization in Sec. VI. We refer to the model with block reordering (Sec. IV-B) and improved normalization as **PointODE**.

### D. PointODE-Elite: Lightweight Version of PointODE

Following PointMLP-Elite, **PointODE-Elite** is introduced as a lightweight version that makes three modifications to PointODE. The feature dimensions $F_1, \ldots, F_4$ are reduced from $(128, 256, 512, 1024)$ to $(64, 128, 256, 256)$. ODEPBlocks employ the bottleneck structure by setting the number of intermediate feature dimensions to $F_s' = F_s/4$ instead of $F_s' = F_s$ as in PointODE (Fig. 4). The output dimensions of the embedding block $F_0$ and the group size $K$ are both halved from 64 to 32, and from 24 to 12, respectively (Fig. 6). With these changes, PointODE-Elite achieves a 8.52x and 9.90x reduction of the parameter size and FLOPs (4.90M to 0.58M, 6.32G to 0.64G), respectively, leading to a 23.02x and 25.51x overall reduction than the baseline PointMLP. The feature extraction part (the embedding block and four stages) only has 0.30M parameters. Even compared to PointMLP-Elite, PointODE-Elite has 1.25x and 1.83x fewer parameters and FLOPs, while achieving the same or slightly higher accuracy.

Considering the unordered nature of point clouds, the network output should be invariant to the ordering of input points [4]. The output of Neural ODE ($\mathbf{h}(t_C)$ in Eq. 2) is permutation invariant given the input feature $\mathbf{h}(t_0)$, if the network $f(\mathbf{h}(t), t, \boldsymbol{\theta})$ is permutation invariant with respect to $\mathbf{h}(t)$ [29]. ODEPBlock satisfies this, because the network layers (i.e., ConcatT, FC, BN and ReLU) apply their respective operations to each $F_s$-dim point feature independently. The other components (i.e., point-wise normalization, FC-BN-ReLU, and max-pooling) are not affected by the input ordering as well, meaning that PointODE is permutation invariant.

## V. FPGA Design and Implementation

In this section, we describe the FPGA design and implementation of PointODE-Elite. We offload the feature extraction part (the embedding block and four stages) to the FPGA, as it accounts for >90% of the entire inference time (Fig. 13).

### A. Point-wise Normalization

Fig. 7 illustrates the block diagram of a stage (except ODEPBlock). The $F_{s-1}$-dim features $\mathcal{F}^{s-1}$ for $N_{s-1}$ points along with the indices of size $(N_s, K)$ are stored on the respective buffers and passed as input to the stage. First, the **Sample+Group** module reads a single row of the index buffer. Based on this, it accesses the feature buffer to collect a feature $\mathbf{f}_j^{s-1} \in \mathcal{F}^{s-1}$ of a group centroid as well as those of the $K$ neighboring points $\{\mathbf{f}_{j,k}^{s-1}\}$ belonging to the same group. The concatenated features $\{[\mathbf{f}_{j,k}^{s-1}, \mathbf{f}_j^{s-1}]\}$ are stored on an intermediate buffer of size $(K, 2F_{s-1})$. Then, the **Mean+Std** module computes a point-wise mean $\{\tilde{\mu}_{j,k}\}$ and deviation $\{\tilde{\sigma}_{j,k}\}$ (Eq. 6) and writes them to another buffer of size $(2, K)$.

The **Transform** module normalizes grouped features (Eq. 5) and the result $\{[\tilde{\mathbf{f}}_{j,k}^{s-1}, \mathbf{f}_j^{s-1}]\}$ is fed to an MLP block consisting of **FC** and **BN-ReLU** modules. Given an input feature $\mathbf{x}$, weight $\mathbf{W}$, and bias $\mathbf{b}$ stored on-chip, **FC** computes $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$. BN and ReLU are fused together into **BN-ReLU**, which computes $\mathbf{y} = \max(\boldsymbol{\gamma}\mathbf{x} + \boldsymbol{\delta}, \mathbf{0})$ based on a scale $\boldsymbol{\gamma}$ and bias $\boldsymbol{\delta}$ stored on-chip. **MaxPool** retrieves $F_s$-dim features of the $K$ grouped points from **BN-ReLU** and aggregates them into a single $F_s$-dim feature via max-pooling, which is stored on an output buffer (Fig. 7, right).

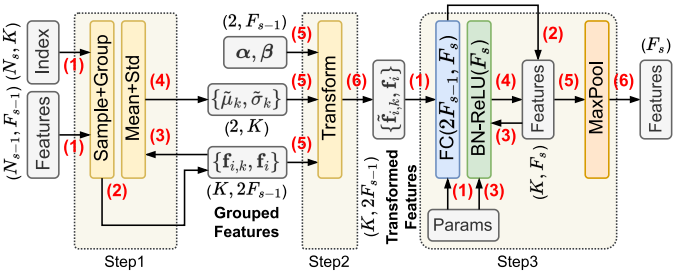


Fig. 7: Block diagram of modules for the point-wise normalization, MLP block (FC-BN-ReLU), and max-pooling.

The above process should be repeated $N_s$ times to extract features for all sampled points in $\mathcal{P}^s$. Since each module

processes a single group at a time, the intermediate buffers do not need to store features for the whole point cloud $\mathcal{P}^s$, which significantly reduces the on-chip memory usage[2].

### B. ODEPBlock

Fig. 8 shows the block diagram of modules for ODEPBlock at stage $s$. An $F_s$-dim aggregated feature is first read from the on-chip buffer and the output buffer is initialized (Fig. 8, right). The ODE forward pass (Eq. 2) is then repeated for $C$ iterations to gradually update the $F_s$-dim output feature. The **ConcatT** module combines a time variable $t \in [t_a, t_b]$ with an input to produce an $(F_s + 1)$-dim augmented feature, which is projected to $F'_s$-dim by **FC** and **BN-ReLU**. The intermediate $F'_s$-dim feature is processed by a sequence of **ConcatT**-**FC**-**BN** modules to obtain an $F_s$-dim output, which is scaled by the step size $h$ and then added to the original input following the Euler numerical integration (Eq. 2). Each module is parallelized by unrolling the loop and partitioning the buffers as needed. Similar to Sec. V-A, the sampled points $\mathcal{P}^s$ (groups) are processed one at a time, which greatly saves the on-chip memory.

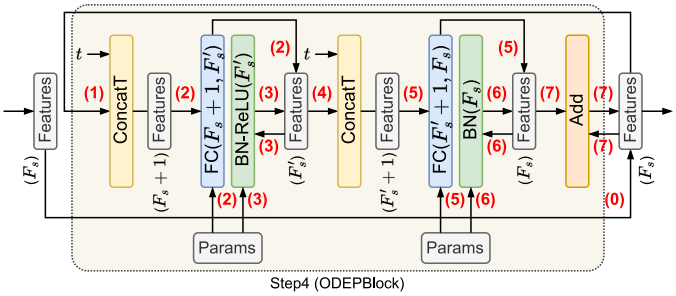


Fig. 8: Block diagram of ODEPBlock.

### C. Implementation of the Stage

The overall architecture of the stage $s$ is depicted in Fig. 9. The modules presented in Sec. V-A–V-B are organized as a four-step pipeline (Fig. 9, bottom), where each step processes a different sampled point in $\mathcal{P}^s$ and its $K$ nearest neighbors. The first step is formed by **Sample+Group** and **Mean+Std** modules, while **Transform** module works as the second step (Fig. 7). The **FC**, **BN-ReLU**, and **MaxPool** modules are tied together to form the third step (Fig. 7), whereas the last step consists of an ODEPBlock (Fig. 8). Since PointODE-Elite incorporates the proposed point-wise normalization (Eqs. 5–6) instead of the geometric affine module (Eqs. 3–4), each sampled point can be processed independently. Unlike PointMLP, each stage can start running the subsequent MLP and ODEPBlock without waiting for the computation of $\sigma$, allowing for pipelined execution. By exploiting these, the total latency of the stage is reduced by 2.83x.

### D. Implementation of PointODE-Elite

Fig. 10 shows an overview of the FPGA implementation. PointODE-Elite is directly mapped onto the FPGA, and the

[2]For instance, the buffer size for normalized features are reduced from $(N_s, K, 2F_{s-1})$ to just $(K, 2F_{s-1})$ by processing a single sampled point (group centroid) at a time.

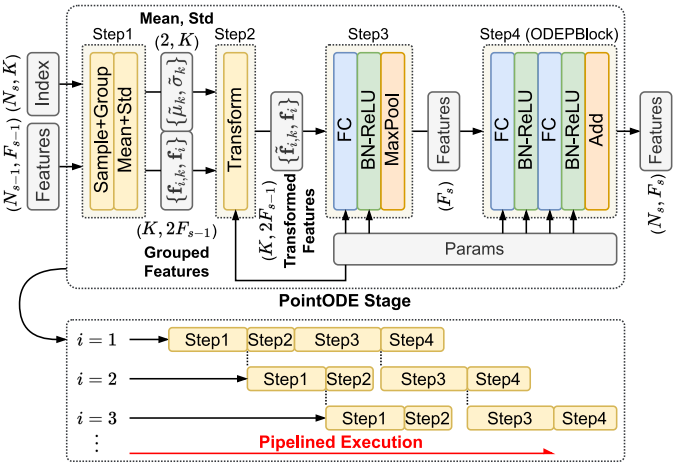


Fig. 9: Block diagram of the stage.

implementation can be divided into four stages as well as an embedding block, with each consisting of a set of modules. The embedding block (Fig. 6) extracts $F_0$-dim features for each of $N$ input points using the **FC** and **BN-ReLU** modules and hands these features off to the first stage.

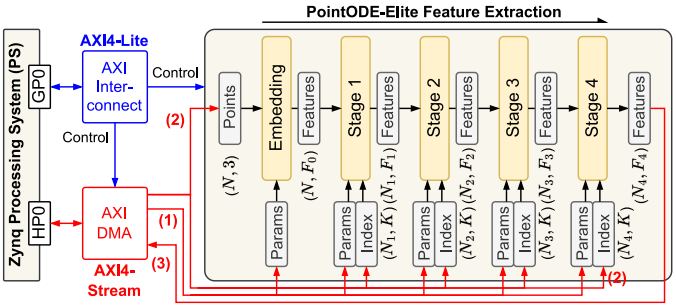


Fig. 10: Overview of the FPGA implementation.

PointODE-Elite is designed as an IP core with two 32-bit AXI interfaces. It uses an AXI4-Stream interface to stream in the input point clouds, parameters, and sampling indices, as well as to stream out the extracted features to DDR. The DMA controller handles the data movement between DDR and FPGA. Besides, the host configures PointODE-Elite and the DMA controller (e.g., $C$ and $h$) through an AXI4-Lite interface. These AXI interfaces are connected to the Processing System (PS) side via high-performance ports.

The whole PointODE-Elite and intermediate buffers fit in the on-chip memory of embedded FPGAs (e.g., ZCU104) thanks to the parameter-efficiency of the network and coarse-grained pipelining within each stage, which substantially reduces the data transfer overhead during inference. The parameters are read from DDR and stored to the respective on-chip buffers (Fig. 10) before inference. At inference, the point cloud $\mathcal{P}$ as well as the sampling indices (of size $(N_s, K)$) are transferred to the on-chip buffers and fed forward to the PointODE-Elite modules. The extracted features (of size $(N_4, F_4)$) are written back to DDR via DMA and used for the specific tasks (e.g., classification). As discussed in Sec. V-A, each stage $s$ needs precomputed indices of size $(N_s, K)$ to subsample $N_s$ out of $N_{s-1}$ points ($\mathcal{P}^s \subset \mathcal{P}^{s-1}$) and find

$K$NNs for each sampled point. The host performs farthest point sampling (FPS) and $K$NN search recursively on the input $\mathcal{P}$ four times to generate indices for $\mathcal{P}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$. In addition to progressive downsampling, the classification network is also executed on the host after receiving extracted features.

## VI. Evaluation

### A. Experimental Setup

*1) Implementation Details:* The proposed design (Fig. 10) is implemented on a Xilinx ZCU104 board. It features 2GB of DDR4 memory and an FPGA device (xczu7ev-2ffvc1156) containing a quad-core ARM Cortex-A53 processor (1.2GHz). Pynq framework is used to write the host code and interact with the FPGA kernel. We implement PointODE-Elite with Vitis HLS 2020.2 and use Vivado 2020.2 to generate a bitstream. The clock frequency of PointODE-Elite and the DMA controller is set to 200MHz. We utilize a 24-bit fixed-point format with an 8-bit integer and a 16-bit fractional part to represent point clouds, extracted features, as well as parameters. PointODE-Elite handles the conversion between fixed- and floating-point values. We apply dataflow optimization to each stage for block-level pipelining (Sec. V-C).

*2) Point Cloud Datasets:* We utilize two popular datasets for point cloud classification: ModelNet40 [38] and ScanObjectNN [39]. ModelNet40 consists of 12311 point clouds across 40 categories (e.g., airplane, table), generated by uniformly sampling the surfaces of synthetic CAD objects. The dataset is split into training and test sets with each containing 9843 and 2468 samples, respectively. ScanObjectNN is a real-world object dataset consisting of 2902 point clouds across 15 categories (e.g., desk, sofa). Unlike ModelNet40, ScanObjectNN is more challenging as its samples are affected by various factors such as measurement error, background noise, and occlusion. We subsample $N = 1024$ points out of 2048. During training, we apply a random scaling of $[0.67, 1.5]$ and a random translation of $[-0.2, 0.2]$ along each axis to point clouds for data augmentation.

*3) Model Training Details:* We train the networks using PyTorch on an Ubuntu 20.04 workstation equipped with Nvidia GeForce RTX 3090 GPUs. The learning rate $\eta$ is initialized to 0.1 and 0.01 for ModelNet40 and ScanObjectNN, respectively[3]. At the training phase, we adopt gradient clipping, a cross-entropy loss, an Adam optimizer with default parameters, and a cosine annealing scheduler with a minimum learning rate of 5e-3. The networks are trained for 300 and 200 epochs with a batch size of 32 and 8 on ModelNet40 and ScanObjectNN, respectively. In Neural ODE (Sec. III-A), the step size $h$ depends on the number of iterations $C$ as well as the integration interval $[t_a, t_b]$. The initial time $t_a$ is fixed at 0, while the final time $t_b$ is chosen from $\{0.1, 0.2, 0.3\}$ for each model and dataset to achieve the best accuracy.

### B. Classification Accuracy

Table I summarizes the classification accuracy (with $N = 1024$ input points), where **mAcc** and **OA** denote the mean class accuracy and overall accuracy[4]. The results for PointMLP and PointODE are obtained on the GPU workstation. By replacing two consecutive ResPBlocks with one ODEPBlock, PointODE-Naive (Sec. IV-A) reduces the parameter size by 1.73x while keeping the accuracy within 0.6–1.6% of PointMLP. Compared to PointMLP, PointODE-Elite consumes 23.02x less parameters and only shows an accuracy drop of 0.1–1.1%, significantly improving a trade-off between model size and performance on both synthetic and real-world datasets. It achieves on-par or even better accuracy with 1.25x fewer parameters than PointMLP-Elite.

While PointODE-Elite is a simple network with residual MLPs and does not require normal information, it outperforms more sophisticated networks with custom convolution kernels [14], [15], [36], graph convolution [17], and transformers [19], [20]. PointODE-Elite surpasses the existing ODE-based method (ExNODE [29] with Set Transformer [40]) by 4.1% on ModelNet40, highlighting the effectiveness of hierarchical feature learning. In addition, it shows competitive performance against state-of-the-art models [23], [24] with >4x fewer parameters. These results demonstrate the effectiveness of Neural ODE-based approach for designing a parameter-efficient network for point clouds.

TABLE I: Classification accuracy.

| Method | ModelNet40 | | ScanObjectNN | | #Param |
|---|---|---|---|---|---|
| | mAcc(%) | OA(%) | mAcc(%) | OA(%) | |
| PointNet [4] | 86.0 | 89.2 | 63.4 | 68.2 | 3.48M |
| PointNet++ [5] | 88.4 | 90.7 | 75.4 | 77.9 | 1.48M |
| PointCNN [14] | 88.1 | 92.5 | 75.1 | 78.5 | – |
| PointConv* [15] | – | 92.5 | – | – | 18.6M |
| KPConv [36] | – | 92.9 | – | – | 14.3M |
| DGCNN [17] | 90.2 | 92.9 | 73.6 | 78.1 | 1.84M |
| ExNODE [29] | – | 89.3 | – | – | 0.52M |
| PAConv [6] | – | 93.9 | – | – | 2.44M |
| PCT [19] | – | 93.2 | – | – | 2.88M |
| PT* [20] | 90.6 | 93.7 | – | – | – |
| RepSurf [23] | 91.4 | 94.4 | 81.3 | 84.3 | 1.48M |
| PointNeXt-S [24] | 90.8 | 93.2 | 85.8 | 87.7 | 1.4M |
| PointMLP | 91.0 | 93.5 | 83.5 | 85.3 | 13.24M |
| PointMLP-Elite | 90.5 | 93.1 | 82.2 | 84.2 | 0.72M |
| PointODE-Naive | 89.4 | 92.9 | 82.5 | 84.3 | 7.66M |
| PointODE-Elite | 90.5 | 93.4 | 82.6 | 84.2 | 0.58M |

Table II presents the ablation results on ScanObjectNN to investigate the impact of the proposed architectural changes. Compared to naively applying Neural ODE (Sec. IV-A), reordering the residual blocks (Sec. IV-B, Fig. 5) allows for 1.56x parameter reduction with an accuracy loss of 0.6%. The feature dimension reduction (Sec. IV-D) leads to 8.52x parameter savings without significantly harming the accuracy. The point-wise normalization (Sec. IV-C) enables the pipelined feature extraction (Fig. 9) and also improves the accuracy by 1.1–1.5% by computing the normalizing factor independently for each point feature.

TABLE II: Ablation study results on ScanObjectNN.

| | Reorder | Point-wise Norm. | Reduced Dims. | ScanObjectNN | |
|---|---|---|---|---|---|
| | | | | mAcc(%) | OA(%) |
| PointODE-Naive | | | | 82.5 | 84.3 |
| | ✓ | | | 81.9 | 83.8 |
| PointODE | ✓ | ✓ | | **83.4** | **84.9** |
| | ✓ | | ✓ | 81.2 | 83.1 |
| PointODE-Elite | ✓ | ✓ | ✓ | 82.6 | 84.2 |

Table III shows the accuracy of FPGA-based PointODE-

[3]We set to $\eta = 0.02$ when training PointODE-Elite on ScanObjectNN.

[4]The asterisk denotes surface normals are used along with point coordinates.

Elite. While the FPGA implementation uses a fixed-point data type for point clouds and extracted features, it maintains nearly the same accuracy as the PyTorch counterpart on both datasets. This suggests more aggressive quantization techniques could be applied to PointODE-Elite, which is left as a future work. Note the reduction of the mean accuracy by 0.5–0.6% indicates the model makes slightly more incorrect predictions for categories with fewer training samples.

TABLE III: Classification accuracy on the ZCU104 board.

| Method | FPGA | ModelNet40 | | ScanObjectNN | |
|---|---|---|---|---|---|
| | | mAcc(%) | OA(%) | mAcc(%) | OA(%) |
| PointODE-Elite | | 90.5 | 93.4 | 82.6 | 84.2 |
| PointODE-Elite | ✓ | 90.0 | 93.7 | 82.0 | 84.3 |

Fig. 11 plots the accuracy of PointODE-Elite with respect to the number of ODE iterations $C$ on ScanObjectNN. By increasing $C$ from 1 to 8, the overall accuracy improves by 1.1% (83.6% to 84.7%) at the cost of linearly increasing computational cost in ODEPBlocks, closing the gap between PointODE-Elite and PointMLP. Setting $C \geq 10$ results in an accuracy drop, which may be attributed to cumulative numerical errors in the ODE solver. Higher-order ODE solvers (e.g., Runge-Kutta) could be used to alleviate this problem.

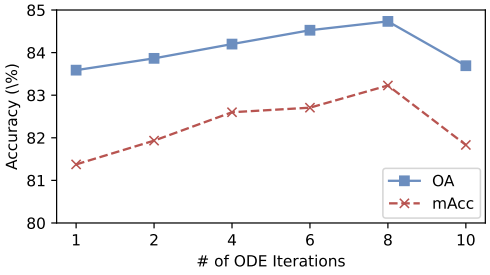


Fig. 11: Number of ODE forward iterations $C$ and accuracy of PointODE-Elite on the ScanObjectNN dataset.

### C. Inference Time

Fig. 12 shows the execution time of the feature extraction part in PointODE-Elite with respect to $C$ and the number of points $N$. Compared to the ARM Cortex-A53 CPU, the FPGA implementation achieves a speedup of 4.39–5.17x and 3.48–5.24x for $C = [1, 8]$ and $N = [512, 2048]$, respectively, thanks to the four-step feature extraction pipeline that processes four points in parallel (Fig. 9). The result indicates the linear computational complexity of ODEPBlocks in terms of both $N$ and $C$. Fig. 13 compares the execution time breakdown of PointODE-Elite ($N = 1024, C = 4$). The feature extraction accounts for 91.7% of the total inference time and dominates the performance. The FPGA implementation accelerates feature extraction by 4.90x (from 337.5ms to 68.8ms), resulting in an overall speedup of 3.70x (from 368.0ms to 99.4ms).

### D. Power and Energy Consumption

The power consumption of PointODE-Elite is calculated by subtracting that of the ZCU104 board in the idle state from that during inference. The power usage is read out from an onboard INA226 sensor via the PMBus interface. We repeat the inference 300 times and average the power

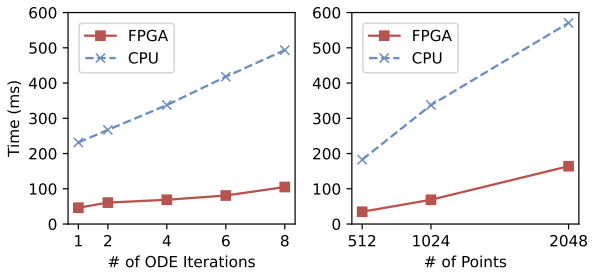


Fig. 12: Time for feature extraction with respect to the number of ODE iterations $C$ (left) and number of points $N$ (right).

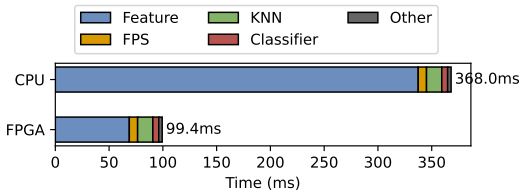


Fig. 13: Execution time breakdown ($N = 1024, C = 4$).

measurements collected at 500ms intervals. PyTorch- and FPGA-based PointODE-Elite consume 0.43W and 0.45W of power (from 11.35W to 11.78W and 11.80W) during inference, respectively. Considering the overall speedup of 3.70x (Sec. VI-C), the FPGA implementation achieves 3.54x higher energy efficiency than the PyTorch counterpart.

### E. FPGA Resource Utilization

Table IV shows the FPGA resource utilization of PointODE-Elite. The design consumes 95% and 69% of the URAM and BRAM slices to store point clouds, extracted features, as well as parameters on-chip, eliminating most of the off-chip memory accesses. The network compression techniques (e.g., quantization and pruning) would further reduce the BRAM and URAM utilization, which in turn allows to assign more memory blocks to each buffer and increase the number of read/write ports. Considering that DSP blocks are underutilized, the design could further parallelize the computation in each module and save the inference time (e.g., each stage can process multiple sampled points at once). More design optimizations should be explored in future work to fully utilize the computational capability of FPGAs.

TABLE IV: FPGA resource utilization.

| | BRAM | URAM | DSP | FF | LUT |
|---|---|---|---|---|---|
| Total | 312 | 96 | 1728 | 460800 | 230400 |
| Used | 215.5 | 92 | 731 | 80871 | 107028 |
| Used (%) | 69.1% | 95.8% | 42.3% | 17.6% | 46.5% |

## VII. Conclusion

In this paper, we propose PointODE as an efficient DNN architecture for point cloud processing, along with its FPGA implementation. PointODE consists of a stack of residual MLP blocks to perform hierarchical feature extraction and aggregation. The key idea is to employ Neural ODE as a network compression technique. We effectively reduce the network size

by replacing a sequence of residual blocks with an ODE-based building block. By adjusting the network architecture, we propose PointODE-Elite as a lightweight version with only 0.58M trainable parameters, which is implemented on a Xilinx ZCU104 board. We design a four-step feature extraction pipeline leveraging point-level parallelism and utilize on-chip memory to eliminate most of the off-chip data transfers. While PointODE-Elite does not employ sophisticated local feature extractors, its FPGA implementation achieves an accuracy of 93.7% on the ModelNet40 dataset, which is even comparable to state-of-the-art networks. In addition, it speeds up the inference by 3.7x and improves the energy efficiency by 3.5x compared to the PyTorch implementation.

**Acknowledgments** This work was supported by JSPS KAKENHI Grant Number 25K24361.

## REFERENCES

[1] J. Park *et al.*, "Colored Point Cloud Registration Revisited," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 143–152.

[2] H. Wang *et al.*, "F-LOAM: Fast LiDAR Odometry and Mapping," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2021, pp. 4390–4396.

[3] T. Yin *et al.*, "Center-Based 3D Object Detection and Tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 11 784–11 793.

[4] C. R. Qi *et al.*, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 652–660.

[5] ——, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, Dec. 2017, pp. 5099–5108.

[6] M. Xu *et al.*, "PAConv: Position Adaptive Convolution With Dynamic Kernel Assembling on Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 3173–3182.

[7] Z.-H. Lin *et al.*, "Learning of 3D Graph Convolution Networks for Point Cloud Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 44, no. 8, pp. 4212–4224, Aug. 2022.

[8] X.-F. Han *et al.*, "Dual Transformer for Point Cloud Analysis," *IEEE Transactions on Multimedia (TMM)*, vol. 25, no. 1, pp. 5638–5648, Aug. 2022.

[9] X. Ma *et al.*, "Rethinking Network Design and Local Geometry in Point Cloud: A Simple Residual MLP Framework," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Apr. 2022, pp. 1–15.

[10] R. T. Q. Chen *et al.*, "Neural Ordinary Differential Equations," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2018, pp. 6571–6583.

[11] Y. Verma *et al.*, "ClimODE: Climate and Weather Forecasting With Physics-informed Neural ODEs," in *Proceedings of the International Conference on Learning Representations (ICLR)*, May 2024, pp. 1–23.

[12] G. Shi *et al.*, "Towards Complex Dynamic Physics System Simulation with Graph Neural Ordinary Equations," *Neural Networks*, vol. 176, no. C, p. 106341, Aug. 2024.

[13] Y. Xu *et al.*, "TiV-ODE: A Neural ODE-based Approach for Controllable Video Generation From Text-Image Pairs," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2024, pp. 14 645–14 652.

[14] Y. Li *et al.*, "PointCNN: Convolution On X-Transformed Points," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2018, pp. 820–830.

[15] W. Wu *et al.*, "PointConv: Deep Convolutional Networks on 3D Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 9621–9630.

[16] Y. Lin *et al.*, "FPConv: Learning Local Flattening for Point Convolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 4293–4302.

[17] Y. Wang *et al.*, "Dynamic Graph CNN for Learning on Point Clouds," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, pp. 1–12, Oct. 2019.

[18] H. Lei *et al.*, "Spherical Kernel for Efficient Graph Convolution on 3D Point Clouds," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 43, no. 10, pp. 3664–3680, Oct. 2021.

[19] M.-H. Guo *et al.*, "PCT: Point Cloud Transformer," *Computational Visual Media*, vol. 7, no. 2, pp. 187–199, Jun. 2021.

[20] H. Zhao *et al.*, "Point Transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 16 259–16 268.

[21] T. Xiang *et al.*, "Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 915–924.

[22] Y. Feng *et al.*, "MeshNet: Mesh Neural Network for 3D Shape Representation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, Jul. 2019, pp. 8279–8286.

[23] H. Ran *et al.*, "Surface Representation for Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 18 942–18 952.

[24] G. Qian *et al.*, "PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2022, pp. 23 192–23 204.

[25] E. Dupont *et al.*, "Augmented Neural ODEs," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2019, pp. 3140–3150.

[26] G.-H. Liu *et al.*, "Second-Order Neural ODE Optimizer," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2021, pp. 25 267–25 279.

[27] A. Gholami *et al.*, "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Aug. 2019, pp. 730–736.

[28] T. Zhang *et al.*, "ANODEV2: A Coupled Neural ODE Framework," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2019, pp. 5151–5161.

[29] Y. Li *et al.*, "Exchangeable Neural ODE for Set Modeling," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2020, pp. 6936–6946.

[30] D. Rempe *et al.*, "CaSPR: Learning Canonical Spatiotemporal Point Cloud Representations," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2020, pp. 13 688–13 701.

[31] H. Kawakami *et al.*, "A Low-Cost Neural ODE with Depthwise Separable Convolution for Edge Domain Adaptation on FPGAs," *IEICE Transactions on Information and Systems*, vol. E106-D, no. 7, pp. 1186–1197, Jul. 2023.

[32] I. Okubo *et al.*, "A Cost-Efficient FPGA-Based CNN-Transformer Using Neural ODE," *IEEE Access*, vol. 12, no. 1, pp. 155 773–155 788, Oct. 2024.

[33] L. Cai *et al.*, "Accelerating Neural-ODE Inference on FPGAs with Two-Stage Structured Pruning and History-based Stepsize Search," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, Feb. 2023, pp. 177–183.

[34] Y. Chen *et al.*, "Hardware Implementation of nmODE on FPGA," in *Proceedings of the International Annual Conference on Complex Systems and Intelligent Science (CSIS-IAC)*, Oct. 2023, pp. 240–246.

[35] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, May 2021, pp. 1–21.

[36] H. Thomas *et al.*, "KPConv: Flexible and Deformable Convolution for Point Clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 6411–6420.

[37] J. L. Ba *et al.*, "Layer Normalization," arXiv Preprint 1607.06450, Jul. 2016.

[38] Z. Wu *et al.*, "3D ShapeNets: A Deep Representation for Volumetric Shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 1912–1920.

[39] M. A. Uy *et al.*, "Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 1588–1597.

[40] J. Lee *et al.*, "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, Jul. 2019, pp. 3744–3753.