POINTODE: LIGHTWEIGHT POINT CLOUD LEARNING WITH NEURAL ORDINARY DIFFERENTIAL EQUATIONS ON EDGE

A PREPRINT

Keisuke Sugiura

Injugacity of Taukuba

University of Tsukuba 1-1-1 Tennôdai, Tsukuba, Ibaraki, Japan sugiura@lila.cs.tsukuba.ac.jp Mizuki Yasuda

Keio University 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan yasuda@arc.ics.keio.ac.jp

Hiroki Matsutani

Keio University 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan matutani@arc.ics.keio.ac.jp

June 12, 2025

ABSTRACT

Embedded edge devices are often used as a computing platform to run real-world point cloud applications, but recent deep learning-based methods may not fit on such devices due to limited resources. 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 MLP blocks. The point-wise normalization is proposed for PointODE to handle the non-uniform distribution of feature points. We introduce PointODE-Elite as a lightweight version with 0.58M trainable parameters and design its dedicated accelerator for embedded FPGAs. The accelerator consists of a four-stage pipeline to parallelize the feature extraction for multiple points and stores the entire parameters on-chip to eliminate most of the off-chip data transfers. 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 the state-of-the-art models on both synthetic and real-world classification datasets, greatly improving the trade-off between accuracy and inference cost.

1 Introduction

Point cloud is a collection of points representing 3D scenes. Thanks to the increasing availability of low-cost 3D scanners (e.g., LiDARs and depth cameras), it serves as a basis for various mobile-edge applications including 3D reconstruction [1,2], mapping [3–6], and object tracking [7,8]. Since PointNet [9] and PointNet++ [10], 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 [11, 12], graph convolutions [13, 14], and transformers [15, 16] to enhance the feature representation, at the cost of increasing model complexity.

The recently-proposed PointMLP [17] takes the opposite direction and avoids the use of complicated feature extractors. PointMLP is composed of a stack of MLP blocks with residual connections, but achieves competitive accuracy to state-of-the-art models on popular tasks such as classification and part segmentation. In addition, PointMLP can

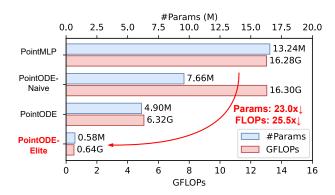


Figure 1: Number of parameters and FLOPs of the proposed PointODE-Elite and the baseline PointMLP.

efficiently leverage the computing power of FPGAs, as it mainly involves highly-parallelizable matrix operations within MLPs. PointMLP is basically a ResNet [18]-like architecture with MLPs instead of 2D convolutions. Compared to convolutions, FC (Fully-Connected) layers require a large number of parameters proportional to the feature dimensions.

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 [19], to represent the hidden dynamics of continuous-time systems. It is widely adopted in a variety of tasks, e.g., time-series forecasting [20–22], trajectory modeling [23, 24], climate forecasting [25, 26], physics simulation [27], video generation [28, 29], and image registration [30]. 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 treated as a powerful model compression technique for ResNet-like models. The resulting model consists of fewer residual blocks that are reused during inference, thereby reducing both logic and memory resource consumptions and making it well-suited to embedded FPGAs. While the complicated design of recent feature extractors has limited the use of Neural ODE, it can be easily applied to PointMLP with minimal architectural changes, thus bridging the research gap between ODEs and point cloud DNNs.

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 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 FPGA 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 the FPGA memory, eliminating most of the off-chip data transfers. 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 Neural ODE-based point cloud analysis.

2 Related Work

DNNs for Point Clouds: PointNet [9] is a pioneering model that directly consumes point clouds. It extracts pointwise features using a shared MLP and aggregates them into a global representation via a symmetric pooling function. PointNet++ [10] is an extension that aims to capture fine geometric context through repeated sampling and grouping

of feature points. PointNet and PointNet++ 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 [11, 12, 31–35], while several works [13, 14, 36–38] employ GCNs (Graph Convolutional Networks) to process KNN graphs built from point clouds. Inspired by the tremendous success, transformer-based methods [15, 16, 39, 40] 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 (sequence of points) [41], polygon meshes [42], or umbrella surfaces [43]. Aside from these approaches, PointNeXt [44] revisit PointNet++ and proposes improved training strategies to boost its performance without changing the network structure. PointMLP [17] opts to use a stack of MLP blocks with residual connections and learns a hierarchical feature representation like PointNet++. PointMLP 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 [19] 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 [45], obtain higher-order derivatives [46], avoid incorrect gradients and numerical instability [47,48], improve robustness to noise [49], and handle graph data [50,51]. ExNODE [52] is a seminal work that extends Neural ODE to point clouds. It uses DeepSet [53] or Set Transformer [54] as a basic component to ensure the permutation invariance. CaSPR [55] uses Neural ODE to learn the representations of dynamically moving point clouds. Exploiting the connection between ResNets [18] and Neural ODEs, the recent work [56,57] 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 [56–59] explore the FPGA design of Neural ODEs. The authors of [56] 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 [57], an FPGA accelerator is proposed for a CNN-Transformer hybrid model combining Neural ODE and ViT (Vision Transformer) [60]. In [58], 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.

3 Background

3.1 Neural ODE

ResNet [18] employs residual (skip, shortcut) connections in its building blocks to stabilize the training of deep networks. A sequence of building blocks (**ResBlock**s) 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 θ_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 $\mathrm{d}\mathbf{h}(t)/\mathrm{d}t = f(\mathbf{h}(t),t,\theta)$, where t denotes a continuous time and $f(\cdot,\theta)$ represents a time-derivative (i.e., dynamics) of a hidden state $\mathbf{h}(t)$. Neural ODE [19] 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:

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

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 θ , 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. Neural ODE can be seen as a technique to compress residual networks by fusing multiple similar ResBlocks into a single ODEBlock.

3.2 PointMLP

PointMLP [17] is a pure MLP-based network with residual connections for point cloud feature extraction. The architecture is presented in Fig. 3. PointMLP directly takes 3D point coordinates $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_N\}$ as input and extracts

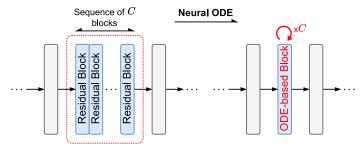


Figure 2: Replacing the forward pass of C ResBlocks with C forward iterations of the single ODE-based building block.

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 $\{\mathbf{f}_1^0, \dots, \mathbf{f}_N^0\}$ for each point, which is followed by a stack of four stages to hierarchically aggregate these local features.

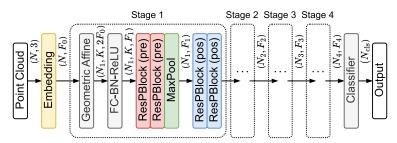


Figure 3: Architecture of PointMLP.

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 (ResPBlocks) 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 [12,33,35,61]). 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.

4 PointODE-Elite

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

4.1 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 number of parameters), this change 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.

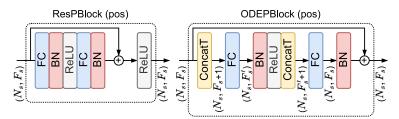


Figure 4: ResPBlock and ODEPBlock.

4.2 Residual Block Reordering

To improve the effectiveness of using ODE blocks, we propose to first reorder the blocks in each stage of PointMLP before employing Neural ODE. 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 two¹. 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.

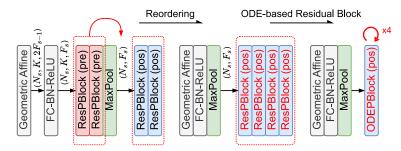


Figure 5: Residual block reordering (left: PointMLP, center: reordered layers, right: PointODE(-Elite)).

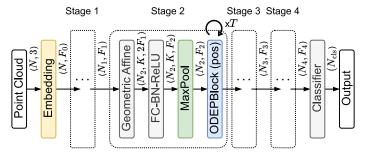


Figure 6: Architecture of PointODE and PointODE-Elite.

¹The first stage keeps two separate ODEPBlocks for accuracy.

4.3 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:

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

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 $\varepsilon = 1\text{e-}5$ is a small constant to avoid zero division. The scalar σ is defined as:

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

Since μ and σ 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 $\mathbf{f} \mapsto \hat{\mathbf{f}}$ is not fully independent for each point. Each stage cannot proceed to the normalization (Eq. 3) and subsequent blocks until σ is available. Besides, the accumulation of $N_{s-1}K$ features requires a wider fixed-point format to avoid overflow Considering these, we propose to compute μ and σ separately for each point in a group (similar to layer normalization [62]) and transform the features $\{\mathbf{f}_{i,k}^{s-1}\}$ as follows:

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

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

where 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. 5). 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. 6. We refer to the model with block reordering (Sec. 4.2) and improved normalization as **PointODE**.

4.4 PointODE-Elite: Lightweight Version of PointODE

Following PointMLP-Elite [17], **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. Notably, 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 [9]. ExNODE [52] proves that the output of Neural ODE ($\mathbf{h}(t_C)$ in Eq. 2) is permutation invariant given the input feature set $\mathbf{h}(t_0)$, if the network $f(\mathbf{h}(t),t,\boldsymbol{\theta})$ is permutation invariant with respect to $\mathbf{h}(t)$. ODEPBlock satisfies this property, 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 maxpooling) are not affected by the ordering of inputs as well, indicating that PointODE produces permutation invariant features.

5 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).

5.1 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} , a weight matrix \mathbf{W} , and a bias vector \mathbf{b} stored on-chip, the **FC** module performs a dot product and an addition $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$. BN and ReLU nonlinearity are fused together into the **BN-ReLU** module, which computes an output $\mathbf{y} = \max(\gamma \mathbf{x} + \delta, \mathbf{0})$ based on a BN scale γ and bias δ stored on-chip. The **MaxPool** module 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). DSP blocks are utilized to compute the statistics $\tilde{\mu}$, $\tilde{\sigma}$ in **Sample+Group**, normalize the grouped features in **Transform**, and perform MAC operations in **FC** as well as **BN-ReLU**.

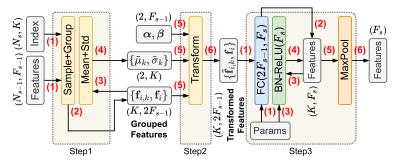


Figure 7: Block diagram of modules for the point-wise normalization, MLP block (FC-BN-ReLU), and max-pooling.

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 , thereby significantly reducing the on-chip memory usage².

5.2 ODEPBlock

Fig. 8 shows the block diagram of modules for ODEPBlock at stage s. ODEPBlock is realized by a set of modules corresponding to the layers and a residual connection (Fig. 4, right), as well as on-chip buffers for storing layer parameters and intermediate results. 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 the **FC** and **BN-ReLU** modules. 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 method (Eq. 2). The operation of each module is parallelized by unrolling the loop and partitioning the buffers as needed. Similar to Sec. 5.1, the sampled points \mathcal{P}^s (groups) are processed one at a time, which greatly saves the on-chip memory.

5.3 Implementation of the Stage

The overall architecture of the stage s is depicted in Fig. 9. The modules presented in Sec. 5.1–5.2 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

²For 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.

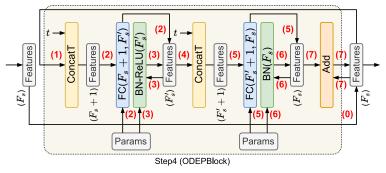


Figure 8: Block diagram of ODEPBlock.

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 σ , which allows for pipelined execution. By exploiting the parallelism of PointODE-Elite and processing four points concurrently, the total latency of the stage is reduced by 2.83x.

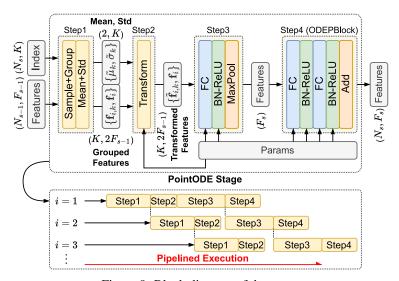


Figure 9: Block diagram of the stage.

5.4 Implementation of PointODE-Elite

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

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 (for FC, BN, and normalization), and sampling indices, as well as to stream out the extracted features to DDR. The DMA controller handles the data movement between DDR and PointODE-Elite. Besides, the host writes configuration parameters of PointODE-Elite and the DMA controller (e.g., the number of ODE iterations C and the ODE step size 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

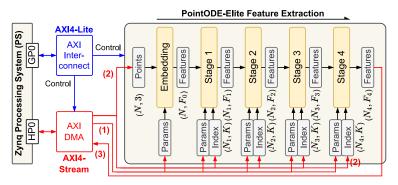


Figure 10: Overview of the FPGA implementation.

in Sec. 5.1, each stage s needs precomputed indices of size (N_s, K) to subsample N_s points \mathcal{P}^s out of N_{s-1} points \mathcal{P}^{s-1} and find KNNs for each sampled point. The host performs farthest point sampling (FPS) and KNN 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.

6 Evaluation

Here we evaluate the performance of PointODE-Elite in terms of (i) accuracy, (ii) inference time, (iii) power consumption, and (iv) resource utilization.

6.1 Experimental Setup

6.1.1 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 clocked at 1.2GHz. The Pynq (Python productivity for Zynq) framework is used to write the host code that interacts with the FPGA kernel. We implement PointODE-Elite using Vitis HLS 2020.2 and then invoke Vivado 2020.2 to generate a bitstream from the block design (Fig. 10). 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 representations. We apply dataflow optimization (provided by Vitis HLS) to each stage for block-level pipelining (Sec. 5.3).

6.1.2 Point Cloud Datasets

We utilize two popular datasets for point cloud classification: ModelNet40 [63] and ScanObjectNN [64]. 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. Each sample is translated and normalized to fit within a unit sphere centered at the origin. 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.

6.1.3 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 η is initialized to 0.1 and 0.01 for ModelNet40 and ScanObjectNN, respectively³. 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. 3.1), the step

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

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.

6.2 Classification Accuracy

Table 1 summarizes the classification accuracy (with N=1024 input points), where **mAcc** and **OA** denote the mean class accuracy and overall accuracy⁴. The results for PointMLP and PointODE are obtained on the GPU workstation. By replacing two consecutive ResPBlocks with one ODEPBlock, PointODE-Naive (Sec. 4.1) 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 [33, 35, 61], graph convolution [36], and transformers [39,40]. PointODE-Elite surpasses the existing ODE-based method (ExNODE [52] with Set Transformer [54]) by 4.1% on ModelNet40, highlighting the effectiveness of hierarchical feature learning. In addition, it shows competitive performance against state-of-the-art models [43,44] with >4x fewer parameters. These results demonstrate the effectiveness of Neural ODE-based approach for designing a parameter-efficient network for point clouds.

Table 1: Classification accuracy.

Method	ModelNet40		ScanObjectNN		4D
	mAcc(%)	OA(%)	mAcc(%)	OA(%)	#Param
PointNet [9]	86.0	89.2	63.4	68.2	3.48M
PointNet++ [10]	88.4	90.7	75.4	77.9	1.48M
PointCNN [33]	88.1	92.5	75.1	78.5	_
PointConv* [35]	_	92.5	_	_	18.6M
KPConv [61]	_	92.9	_	_	14.3M
DGCNN [36]	90.2	92.9	73.6	78.1	1.84M
ExNODE [52]	_	89.3	_	_	0.52M
PAConv [12]	_	93.9	_	_	2.44M
PCT [39]	_	93.2	_	_	2.88M
PT* [40]	90.6	93.7	_	_	_
RepSurf [43]	91.4	94.4	81.3	84.3	1.48M
PointNeXt-S [44]	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 2 presents the ablation results on ScanObjectNN to investigate the impact of the proposed architectural changes. Compared to naively applying Neural ODE (Sec. 4.1), reordering the residual blocks (Sec. 4.2, Fig. 5) allows for 1.56x parameter reduction with an accuracy loss of 0.6%. The feature dimension reduction (Sec. 4.4) leads to 8.52x parameter savings without significantly harming the accuracy. The point-wise normalization (Sec. 4.3) 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 2: Ablation study results on ScanObjectNN.

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

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

Table 3 shows the accuracy of FPGA-based PointODE-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 3: 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	\checkmark	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, which closes the gap between PointODE-Elite and PointMLP. Setting $C \geq 10$ results in a sudden 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 instead to alleviate this problem.

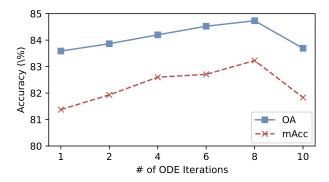


Figure 11: Number of ODE forward iterations C and accuracy of PointODE-Elite on the ScanObjectNN dataset.

6.3 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).

6.4 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 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. 6.3), the FPGA implementation achieves 3.54x higher energy efficiency than the PyTorch counterpart.

6.5 FPGA Resource Utilization

Table 4 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

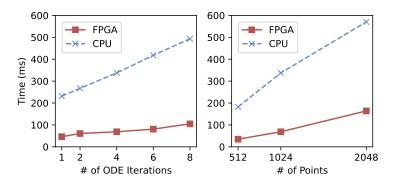


Figure 12: Time for feature extraction with respect to the number of ODE iterations C (left) and number of points N(right).

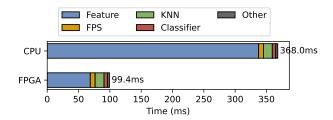


Figure 13: Execution time breakdown (N = 1024, C = 4).

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.

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

Table 4: FPGA resource utilization.

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 behind PointODE 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.

References

[1] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust Reconstruction of Indoor Scenes. In *Proceedings of* the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 5556–5565, June 2015.

- [2] Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Colored Point Cloud Registration Revisited. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 143–152, October 2017.
- [3] Ji Zhang and Sanjiv Singh. LOAM: Lidar Odometry and Mapping in Real-time. In *Proceedings of the Robotics: Science and Systems (RSS)*, pages 1–9, July 2014.
- [4] Tixiao Shan and Brendan Englot. LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765, October 2018.
- [5] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. LIO-SAM: Tightly-coupled Lidar Intertial Odometry via Smoothing and Mapping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142, October 2020.
- [6] Han Wang, Chen Wang, Chun-Lin Chen, and Lihua Xie. F-LOAM: Fast LiDAR Odometry and Mapping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4390–4396, September 2021.
- [7] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking Objects as Points. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 474–490, August 2020.
- [8] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-Based 3D Object Detection and Tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11784–11793, June 2021.
- [9] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, July 2017.
- [10] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 5099–5108, December 2017.
- [11] Yiqun Lin, Zizheng Yan, Haibin Huang, Dong Du, Ligang Liu, Shuguang Cui, and Xiaoguang Han. FPConv: Learning Local Flattening for Point Convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4293–4302, June 2020.
- [12] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. 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)*, pages 3173–3182, June 2021.
- [13] Haoran Zhou, Yidan Feng, Mingsheng Fang, Mingqiang Wei, Jing Qin, and Tong Lu. Adaptive Graph Convolution for Point Cloud Analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4965–4974, October 2021.
- [14] Zhi-Hao Lin, Sheng-Yu Huang, and Yu-Chiang Frank Wang. Learning of 3D Graph Convolution Networks for Point Cloud Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(8):4212–4224, August 2022.
- [15] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-BERT: Pre-Training 3D Point Cloud Transformers With Masked Point Modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19313–19322, June 2022.
- [16] Xian-Feng Han, Yi-Fei Jin, Hui-Xian Cheng, and Guo-Qiang Xiao. Dual Transformer for Point Cloud Analysis. *IEEE Transactions on Multimedia (TMM)*, 25(1):5638–5648, August 2022.
- [17] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking Network Design and Local Geometry in Point Cloud: A Simple Residual MLP Framework. In *Proceedings of the International Conference on Learning Representations (ICLR)*, pages 1–15, April 2022.
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, June 2016.
- [19] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural Ordinary Differential Equations. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 6571–6583, December 2018.
- [20] Yulia Rubanova, Ricky T. Q. Chen, and David K. Duvenaud. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Proceedings of the Advances in Neural Information Processing Systems* (NeurIPS), pages 5320–5330, December 2019.

- [21] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 753–763, August 2020.
- [22] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural Controlled Differential Equations for Irregular Time Series. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 6696–6707, December 2020.
- [23] Yuxuan Liang, Kun Ouyang, Hanshu Yan, Yiwei Wang, Zekun Tong, and Roger Zimmermann. Modeling Trajectories with Neural Ordinary Differential Equations. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1498–1504, August 2021.
- [24] Song Wen, Hao Wang, and Dimitris Metaxas. Social ODE: Multi-agent Trajectory Forecasting with Neural Ordinary Differential Equations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 217–233, October 2022.
- [25] Jeehyun Hwang, Jeongwhan Choi, Hwangyong Choi, Kookjin Lee, Dongeun Lee, and Noseong Park. Climate Modeling with Neural Diffusion Equations. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 230–239, December 2021.
- [26] Yogesh Verma, Markus Heinonen, and Vikas Garg. ClimODE: Climate and Weather Forecasting With Physics-informed Neural ODEs. In *Proceedings of the International Conference on Learning Representations (ICLR)*, pages 1–23, May 2024.
- [27] Guangsi Shi, Daokun Zhang, Ming Jin, Shirui Pan, and Philip S. Yu. Towards Complex Dynamic Physics System Simulation with Graph Neural Ordinary Equations. *Neural Networks*, 176(C):106341, August 2024.
- [28] Sunghyun Park, Kangyeol Kim, Junsoo Lee, Jaegul Choo, Joonseok Lee, Sookyung Kim, and Edward Choi. Vid-ODE: Continuous-Time Video Generation with Neural Ordinary Differential Equation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2412–2422, February 2021.
- [29] Yucheng Xu, Nanbo Li, Arushi Goel, Zonghai Yao, Zijian Guo, and Hamidreza Kasaei. TiV-ODE: A Neural ODE-based Approach for Controllable Video Generation From Text-Image Pairs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 14645–14652, May 2024.
- [30] Yifan Wu, Tom Z. Jiahao, Jiancong Wang, Paul A. Yushkevich, M. Ani Hsieh, and James C. Gee. NODEO: A Neural Ordinary Differential Equation Based Optimization Framework for Deformable Image Registration. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 20804– 20813, June 2022.
- [31] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point Convolutional Neural Networks by Extension Operators. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, August 2018.
- [32] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, September 2018.
- [33] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution On X-Transformed Points. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 820–830, December 2018.
- [34] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-Shape Convolutional Neural Network for Point Cloud Analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8895–8904, June 2019.
- [35] Wenxuan Wu, Zhongang Qi, and Li Fuxin. PointConv: Deep Convolutional Networks on 3D Point Clouds. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 9621– 9630, June 2019.
- [36] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, October 2019.
- [37] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. Grid-GCN for Fast and Scalable Point Cloud Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5661–5670, June 2020.
- [38] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical Kernel for Efficient Graph Convolution on 3D Point Clouds. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 43(10):3664–3680, October 2021.

- [39] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R. Martin, and Shi-Min Hu. PCT: Point Cloud Transformer. *Computational Visual Media*, 7(2):187–199, June 2021.
- [40] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H.S. Torr, and Vladlen Koltun. Point Transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16259–16268, October 2021.
- [41] Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 915–924, October 2021.
- [42] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. MeshNet: Mesh Neural Network for 3D Shape Representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8279–8286, July 2019.
- [43] Haoxi Ran, Jun Liu, and Chengjie Wang. Surface Representation for Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18942–18952, June 2022.
- [44] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 23192–23204, December 2022.
- [45] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 3140–3150, December 2019.
- [46] Guan-Horng Liu, Tianrong Chen, and Evangelos Theodorou. Second-Order Neural ODE Optimizer. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), pages 25267–25279, December 2021.
- [47] Amir Gholami, Kurt Keutzer, and George Biros. ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 730–736, August 2019.
- [48] Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E Gonzalez, Kurt Keutzer, Michael W. Mahoney, and George Biros. ANODEV2: A Coupled Neural ODE Framework. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 5151–5161, December 2019.
- [49] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural SDE: Stabilizing Neural ODE Networks with Stochastic Noise. arXiv Preprint 1906.02355, June 2019.
- [50] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph Neural Ordinary Differential Equations. arXiv Preprint 1911.07532, November 2019.
- [51] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Continuous Graph Neural Networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 10432–10441, July 2020.
- [52] Yang Li, Haidong Yi, Christopher Bender, Siyuan Shan, and Junier B Oliva. Exchangeable Neural ODE for Set Modeling. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 6936–6946, December 2020.
- [53] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep Sets. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 3391–3401, December 2017.
- [54] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3744–3753, July 2019.
- [55] Davis Rempe, Tolga Birdal, Yongheng Zhao, Zan Gojcic, Srinath Sridhar, and Leonidas Guibas. CaSPR: Learning Canonical Spatiotemporal Point Cloud Representations. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 13688–13701, December 2020.
- [56] Hiroki Kawakami, Hirohisa Watanabe, Keisuke Sugiura, and Hiroki Matsutani. A Low-Cost Neural ODE with Depthwise Separable Convolution for Edge Domain Adaptation on FPGAs. *IEICE Transactions on Information and Systems*, E106-D(7):1186–1197, July 2023.
- [57] Ikumi Okubo, Keisuke Sugiura, and Hiroki Matsutani. A Cost-Efficient FPGA-Based CNN-Transformer Using Neural ODE. *IEEE Access*, 12(1):155773–155788, October 2024.
- [58] Lei Cai, Jing Wang, Lianfeng Yu, Bonan Yan, Yaoyu Tao, and Yuchao Yang. 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)*, pages 177–183, February 2023.

- [59] Yi Chen, Hanwen Liu, Enqi Zhang, Hong Qu, and Zhang Yi. Hardware Implementation of nmODE on FPGA. In *Proceedings of the International Annual Conference on Complex Systems and Intelligent Science (CSIS-IAC)*, pages 240–246, October 2023.
- [60] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proceedings of the International Conference on Learning Representations (ICLR)*, pages 1–21, May 2021.
- [61] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, Francois Goulette, and Leonidas J. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6411–6420, October 2019.
- [62] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. arXiv Preprint 1607.06450, July 2016.
- [63] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, June 2015.
- [64] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. 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)*, pages 1588–1597, October 2019.