ADVECTIVENET: AN EULERIAN-LAGRANGIAN FLUIDIC RESERVOIR FOR POINT CLOUD PROCESSING

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper presents a novel physics-inspired deep learning approach for point cloud processing motivated by the natural flow phenomena in fluid mechanics. Our learning architecture jointly defines data in an Eulerian world space, using a static background grid, and a Lagrangian material space, using moving particles. By introducing this Eulerian-Lagrangian representation, we are able to naturally evolve and accumulate particle features using flow velocities generated from a generalized, high-dimensional force field. We demonstrate the efficacy of this system by solving various point cloud classification and segmentation problems with state-of-the-art performance. The entire geometric reservoir and data flow mimics the pipeline of the classic PIC/FLIP scheme in modeling natural flow, bridging the disciplines of geometric machine learning and physical simulation.

1 INTRODUCTION

The fundamental insight of deep learning is to discover feature representations from complex data sets using a hierarchical model composed of layers of simpler data structures. These data structures, such as a uniform grid (Lecun et al., 1998), an unstructured graph (Kipf & Welling, 2016), or a hierarchical point set (Qi et al., 2016a; 2017), function as a geometric reservoir to yield intricate underpinning patterns by evolving the massive input data in a high-dimensional parameter space. On another front, computational physics researchers have been mastering the art of inventing geometric data structures and simulation algorithms to model complex physical systems. Lagrangian structures, which track the motion in a moving local frame such as a particle system (Monaghan, 1992), and Eulerian structures, which describe the evolution in a fixed world frame such as a Cartersian grid (Foster & Fedkiw, 2001; Fedkiw et al., 2001), are the two mainstream approaches. Various differential operators have been devised on top of these data structures to model complex fluid or solid systems.

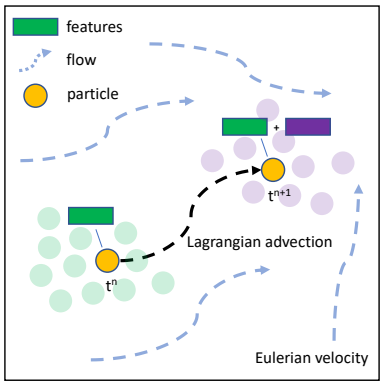

Figure 1: We build an advective network to create a fluidic reservoir with hybrid Eulerian-Lagrangian representations for point cloud processing.

Pioneered by E (2017) and popularized by many others, e.g., (Long et al., 2018; Chen et al., 2018; Ruthotto & Haber, 2018), treating the data flow as the evolution of a dynamic system is connecting machine learning and physics simulation. As E (2017) notes, there exists a mathematical equivalence between the forward data propagation on a neural network and the temporal evolution of a dynamic system. Accordingly, the training process of a neural network amounts to finding the optimal control forces exerted on a dynamic system to minimize a specific energy form.

Point cloud processing is of particular interest under this perspective. The two main challenges: to build effective convolution stencils and to evolve learned nonlinear features (Qi et al., 2016a; Atzmon et al., 2018; Wang et al., 2019), can map conceptually to the challenges of devising world-frame differential operators and tracking material-space continuum deformations when simulating

a PDE-driven dynamic system in computational physics. We envision that the key to solving these challenges lies in the adaption of the most suited geometric data structures to synergistically handle the Eulerian and Lagrangian aspects of the problem. In particular, it is essential to devise data structures and computational paradigms that can accommodate global fast convolutions, and at the same time track non-linear feature evaluations.

The key motivation of this work originates from physical computing that tackles its various frame-dependent and temporally-evolved computational challenges by creating the most natural and effective geometric toolsets under the two different viewpoints. We are specifically interested in uncovering the intrinsic connections between a point cloud learning problem and a computational fluid dynamic (CFD) problem. We observe that the two problems share an important common thread regarding their computational model, which both evolve Lagrangian particles in an Eulerian space guided by the first principle of energy minimization. Such observations shed new insight into the 3D point cloud processing and further opens the door for marrying the state-of-the-art CFD techniques to tackle the challenges emerging in point cloud learning.

To this end, this paper conducts a preliminary exploration to establish an Eulerian-Lagrangian fluidic reservoir that accommodates the learning process of point clouds. The key idea of the proposed method is to solve the point cloud learning problem as a flow *advection* problem jointly defined in a *Eulerian* world space and a *Lagrangian* material space. The defining characteristic distinguishing our method from others is that the spatial interactions among the Lagrangian particles can evolve temporally via *advection* in a learned flow field, like their fluidic counterpart in a physical circumstance. This inherently takes advantage of the fundamental flow phenomena in evolving and separating Lagrangian features non-linearly (see Figure 1). In particular, we draw the idea of Lagrangian advection on an Eulerian reservoir from both the Particle-In-Cell (PIC) method (Evans & Harlow, 1957) and the Fluid-Implicit-Particle (FLIP) method (Brackbill et al., 1987), which are wholly recognized as 'PIC/FLIP' in modeling large-scale flow phenomena in both computational fluids, solids, and even visual effects. We demonstrate the result of this synergy by building a physics-inspired learning pipeline with straightforward implementation and matching the state-of-the-art with this framework.

The key contributions of our work include:

- An advective scheme to mimic the natural flow convection process for feature separation;
- A fluid-inspired learning paradigm with effective particle-grid transfer schemes;
- A fully Eulerian-Lagrangian approach to process point clouds, with the inherent advantages in creating Eulerian differential stencils and tracking Lagrangian evolution;
- A simple and efficient physical reservoir learning algorithm.

2 Related Works

This section briefly reviews the recent related work on point cloud processing. According to data structures used for building the convolution stencil, the methods can be categorized as Lagrangian (using particles only), Eulerian (using a background grid), and hybrid (using both). We also review the physical reservoir methods that embed network training into a physical simulation process.

Lagrangian Lagrangian methods build convolution operators on the basis of local points. Examples include PointNet (Qi et al., 2016a), which conducts max pooling to combat any disorganized points, PointNet++ (Qi et al., 2017), which leverages farthest point sampling to group particles, and a set of work (Wang et al., 2019; Xu et al., 2018; Li et al., 2018b;a; Jiang et al., 2018) based on k-nearest neighbors. Beyond the mesh-free approaches, researchers also seek to build effective point-based stencils by establishing local connectivities among points. Most significantly, geometric deep learning (Bruna et al., 2013; Bronstein et al., 2016) builds convolution operators on top of a mesh to uncover the intrinsic features of objects' geometry. In particular, we want to highlight the work on dynamic graph CNN (Wang et al., 2019), which builds directed graphs in an extemporaneous fashion in feature space to guide the point neighbor search process, which shares similarities with our approach.

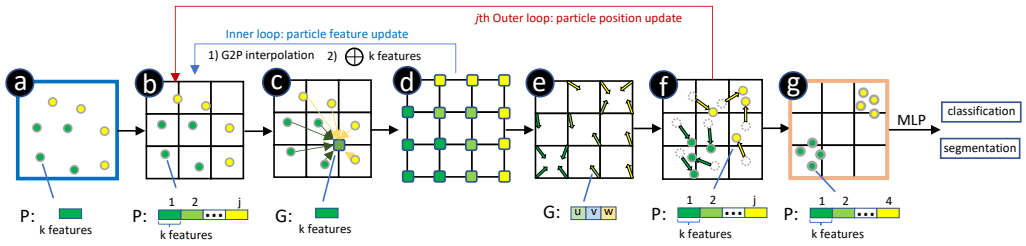

Figure 2: **Workflow overview:** a) The feature vector for each particle is initialized by a 1×1 convolution; b) Particles are embedded in an Eulerian grid; c) Features are interpolated from particles to the grid (P2G); d) 3D convolution is applied on the grid to calculate generalized forces; e) A velocity field is generated on the background grid; f) Particles advect in the Eulerian space using the interpolated velocities; g) Particles aggregate. The workflow exhibits two loops: First, grid features are interpolated to particles (G2P) and appended to its feature vector (inner loop, blue arrow). Second, particle positions are updated for the next time step (outer loop, red arrow). Finally, the Lagrangian features are fed into a fully-connected network for classification and segmentation.

Eulerian Eulerian approaches leverage background discretizations to perform computation. The most successful Eulerian method is the CNN (Lecun et al., 1998), which builds the convolution operator on a 2D uniform grid. This Eulerian representation can be used to process 3D data by using multiple views (Su et al., 2015; Qi et al., 2016b; Feng et al., 2018) and extended to 3D volumetric grids (Maturana & Scherer, 2015; Qi et al., 2016b; Z. Wu, 2015). Grid resolution is the main performance bottleneck for 3D CNN methods. Adaptive data structures such as Octree (Riegler et al., 2016; Wang et al., 2017) and Kd-tree (Klokov & Lempitsky, 2017) were invented to alleviate the problem. Another example of Eulerian structures is Spherical CNN (Cohen et al., 2018) that projects 3D shapes onto a spherical coordinate system to define equivalent rotation convolution.

Hybrid There have been recent attempts to transfer data between Lagrangian and Eulerian representations for efficient convolution implementation. These data transfer methods can be one-way (Wang et al., 2017; Klokov & Lempitsky, 2017; Tchapmi et al., 2017; Le & Duan, 2018), in which case the data is mapped from points to grid cells permanently, or two-way (Su et al., 2018; Atzmon et al., 2018; Liu et al., 2019), in which case data is pushed forward from particle to grid for convolution and pushed backward from grid to particle for evolution. Auto-encoders on point clouds (Fan et al., 2016; Achlioptas et al., 2017; Yang et al., 2017; Yu et al., 2018) can be also regarded as a hybrid approach, where encoded data is Eulerian and decoded data is Lagrangian. In addition, we want to mention the physical reservoir computing techniques that focus on the leverage of the temporal, physical evolution to solve learning problems, e.g., see (Jaeger, 2001) and (Maass et al., 2002). Physical reservoir computing is demonstrating successes in various applications (Jalalvand et al., 2015; Jaeger, 2002; Hauser et al., 2012; Lukoeviius & Jaeger, 2009; Tanaka et al., 2019).

3 ALGORITHM

PIC/FLIP overview Before describing the details of our method, we begin with briefly surveying the background of the PIC/FLIP method. PIC/FLIP uses a hybrid grid-particle representation to describe fluid evolution. The particles are used for tracking materials, and the grid is used for discretizing space. Properties such as mass, density, and velocity are carried on particles. Each simulation step consists of four substeps: particle-to-grid transfer (*P2G*), grid force calculation (*Projection*), grid-to-particle transfer (*G2P*), and moving particles (*advection*). In the *P2G* step, the properties on each particle are interpolated onto a background grid. In the *Projection* step, calculations such as adding body forces and enforcing incompressibility are conducted on the background grid. After this, the velocities on grid nodes are interpolated back onto particles (*G2P*). Finally, particles move to their new positions for the next time step using the updated velocities (*Advection*). As summarized above, the key philosophy of PIC/FLIP is to carry all features on particles and to perform all differential calculations on the grid. The background grid functions as a computational paradigm

that can be established extemporaneously when needed. Data will transfer from particle to grid and then back to particle to finish a simulation loop.

Our proposed approach follows the same design philosophy as PIC/FLIP by storing the learned features on particles and conducting differential calculations on the grid. The Lagrangian features will evolve with the particles moving in an Eulerian space and interact with local grid nodes. As shown in Figure 2, the learning pipeline mimics the PIC/FLIP simulation loop in the sense that Lagrangian particles are advected passively in an Eulerian space guided by a learned velocity field.

Initialization We initialize a particle system $\boldsymbol{P}$ and a background grid $\boldsymbol{G}$ as the Lagrangian and Eulerian representations respectively for processing point clouds. We use the subscript p to refer to particle indices and i to refer to the grid nodes. For the Lagrangian portion, the particle system has n particles, with each particle P_p carrying its position $\boldsymbol{x}_p \in \mathbb{R}^3$, velocity $\boldsymbol{v}_p \in \mathbb{R}^3$, mass $m_p \in \mathbb{R}$, and a feature vector $\boldsymbol{f}_p \in \mathbb{R}^k$ ($k = 64$ initially). The particle velocity is zero at the beginning. The particle mass $m_p = 1$ will keep constant over the entire evolution. The feature vector $\boldsymbol{f}_p$ is initialized by feeding $\boldsymbol{x}_p$ into a multi-layer perceptron (MLP) to extend the dimension.

For the Eulerian part, we start with a 3D uniform grid $\boldsymbol{G}$ to represent the bounding box of the particles. The resolution of the grid is N^3 ($N = 16$ for most of our cases). At the beginning, the particle system and its bounding box are normalized to the space of $[-1, 1]^3$. Each grid node G_i of $\boldsymbol{G}$ stores data interpolated from the particles.

Particle-to-grid transfer The data flow begins with a transfer of properties from particles to grid nodes according to the mass interpolation

$$m_i = \sum_p \omega_{ip} m_p, \tag{1}$$

and the feature interpolation

$$\boldsymbol{f}_i = \sum_p \omega_{ip} m_p \boldsymbol{f}_p / m_i, \tag{2}$$

with the transfer weight ω_{ip} calculated using an interpolation function $\omega_{ip} = P(\boldsymbol{x}_i - \boldsymbol{x}_p)$. In our implementation, we chose to use the Gaussian interpolation for P as:

$$w_{ip} = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_p\|^2 / 2\sigma^2} \tag{3}$$

with σ as the standard deviation for the Gaussian kernel. We pick $\sigma = 1/N$ to concentrate the particle contribution within one grid cell.

Generalized grid forces With the feature vectors transferred from particles to grid nodes, we devise a 3D CNN on the grid to calculate a generalized force field based on the Eulerian features. The network consists of three convolution layers, with each layer as a combination of 3D convolution, batch norm, and ReLU. The input of the network is a vector field $\mathbf{F}^{(kj)\times N\times N\times N}$ composed of the feature vectors on all grid nodes, with k as the feature vector size (64 by default) and j as the iteration index in the evolution loop (see Figure 2). The output is a convoluted vector field $\mathbf{F}_c^{(kj)\times N\times N\times N}$ with the same size as $\mathbf{F}$.

We use $\mathbf{F}_c$ for two purposes: 1) To interpolate $\mathbf{F}_c$ from the grid back onto particles and append it to the current feature vector in order to enrich its feature description (see G2P and Figure 2 ($d \to b$) for details); 2) To feed $\mathbf{F}_c$ into another single-layer network to generate the new Eulerian velocity field $\mathbf{V}$ for the particle advection. Specifically, this $\mathbf{V}$ is interpolated back onto particles in the same way as the feature interpolation to update the particle positions for the next iteration (see Advection for details).

Grid-to-particle transfer The interpolation from grid to particles is executed using tri-linear interpolation, which is a common scheme for property transfer in simulation and learning code.

Advection The essence of an advection process is to solve the advection equation with the Lagrangian form $D\boldsymbol{v}/Dt = 0$ or the Eulerian form $\partial \boldsymbol{v}/\partial t + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = 0$. The advection equation describes the passive evolution of particle properties within a flow field. With the learned grid velocity

field in hand, we will update the particle velocity following the conventional scheme of PIC/FLIP. Specifically, the new velocity is first updated by interpolating the Eulerian velocity to particles (the PIC step):

$$\boldsymbol{v}_{PIC}^{n+1} = G2P(\boldsymbol{v}_g^{n+1}) \tag{4}$$

Then, we interpolate the difference between the new and the old Eulerian velocity:

$$\boldsymbol{v}_{FLIP}^{n+1} = \boldsymbol{v}_p^n + G2P(\boldsymbol{v}_g^{n+1} - P2G(\boldsymbol{v}_p^n)), \tag{5}$$

and then add them to the particle with a weight α (=0.5 in default.):

$$\boldsymbol{v}_p^{n+1} = \alpha * \boldsymbol{v}_{PIC}^{n+1} + (1-\alpha) * \boldsymbol{v}_{FLIP}^{n+1} \tag{6}$$

With the updated velocity on each particle from the G2P interpolation, the particle's position for the next time step can be updated using a standard time integration scheme (explicit Euler in our implementation):

$$\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \boldsymbol{v}_p^{n+1}\Delta t. \tag{7}$$

Boundary conditions We apply a soft boundary constraints by adding an penalty term in the objective function to avoid particles moving outside of the grid:

$$\phi_b = \frac{1}{n}\sum_p max(0, \|x_p\|_2 - 1) \tag{8}$$

where x_p represents the pth particle in the whole batch and n is the number of particles in the whole batch. We penalize on all the particles that run outside the grid.

We also design gather penalty and diffusion objectives to enhance the particle diffusion and clustering effects during the evolution (specifically for the segmentation application):

$$\phi_g = \frac{1}{2}\sum_l \sum_m max(0, 1 - \|c_l - c_m\|) \tag{9}$$

$$\phi_d = \frac{1}{n}\sum_l \sum_p \|c_l - x_{lp}\| \tag{10}$$

where c_l and c_m are the centers of particles of label l and m and x_{lp} is the pth particle with label l.

4 Network Architecture

The global architecture of our network is shown in Figure 3. Our model starts from a point cloud with position information on each point. After an MLP (32,32,64), each point carries a feature vector of length 64. These features are fed into an interpolation module to exchange information with neighbors. The generated features mainly have two uses: the first is to generate velocity for each particle through the advection module; the second is to be used along with the new advected particle position to collect information from neighbors by another interpolation module. This interpolation-advection process repeats for a certain number of times to accumulate features in the feature space and to aggregate particles in the physical space.

Interpolation module The data flow inside the interpolation module starts with particles, passes through layers of grids, then sinks back to particles. The workflow follows the sequence of $P2G \rightarrow Conv \rightarrow G2P \rightarrow Concatenation$. This module takes the position and the feature vector as input. The feature vectors are first fed into an MLP to reduce its dimensions to 32, which saves computational time and prevents over-fitting. Then, we apply three layers of convolution that are each a combination of 3D convolution, batch norm, and ReLU, with a hidden-layer size as (32,16,32) on the grid. Afterwards, the input and output features (with 32-dimension each) are concatenated together and appended to the original feature vector.

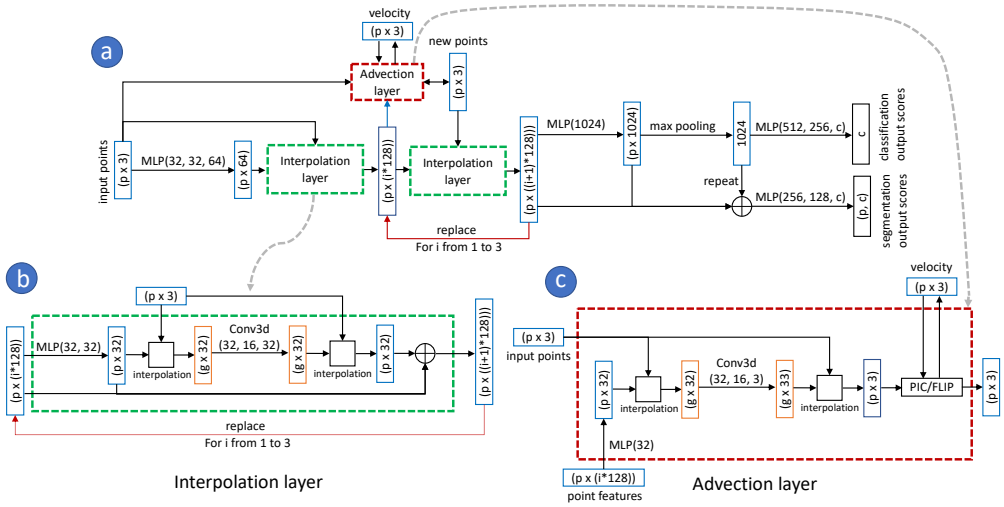

Figure 3: **Network architectures:** a) The top diagram demonstrates the global architecture of our Eulerian-Lagrangian fluidic reservoir network with detailed information for tensor dimensionality and modular connectivity. The blue box is for particle states and the orange box is for grid states. The dot green box and the dot red box are for the functional modules of interpolation and advection with more details illustrated in b) and c). The states are connected with multi-layer perceptrons (MLP-marked arrows in the diagram). Each MLP has a number of hidden layers with a different number of neurons (specified by the numbers within the parentheses). b) The bottom left figure shows the details of the interpolation module updating the particle *features* by transferring data on the grid and concatenating on the particles. c) The bottom right figure shows the advection module which updates the particle *positions* with the generalized Eulerian forces calculated on the grid.

Advection module The data flows from particles, to grid, to particles in the advection module following the similar direction as in the interpolation module (see Figure 3 bottom-right and Figure 2 inner-loop). The primary difference lies in the update of particle positions: the neighboring relations between the grid cells and the moving particles are changed after each advection step. Similar to the interpolation module, particle features are reduced to 32-dimension by an MLP first. Then, these features are interpolated onto the grid following a two-layer convolution (32, 16) to obtain a high-dimensional, generalized force field on the grid. A velocity field is generated from this force field by another single-layer network. The velocity field is then interpolated back to particles for Lagrangian advection. The output of the advection module is a set of particles with new positions that is ready to process for the next outer loop as in Figure 2.

5 Experiments

We conducted three parts of experiments, including the ablation tests and the applications for classification and segmentation. We implemented the system in PyTorch (see the submitted source code) and conducted all the tests on a single RTX 2080 Ti GPU. In the ablation tests, we evaluated the functions of the advection module, grid resolution, and number of points on ShapeNet (Yi et al., 2016) and ModelNet40 (Z. Wu, 2015). To make the ablation test on ShapeNet simple, we only use the point positions for training and testing and we train one network for all objects. The main results are illustrated in Figure 4. For classification, we tested our network on ModelNet40 and its subset ModelNet10. We used the class prediction accuracy as our metric. For segmentation, we tested our network on ShapeNet (Yi et al., 2016) and S3DIS data set (Armeni et al., 2016). We used mean Intersection over Union (mIoU) to evaluate our method and compare with other benchmarks.

5.1 Ablation Experiments

Advection To verify the effectiveness of the advection module, we turn off these modules by replacing them with the interpolation modules for placeholder. We conducted the comparison on

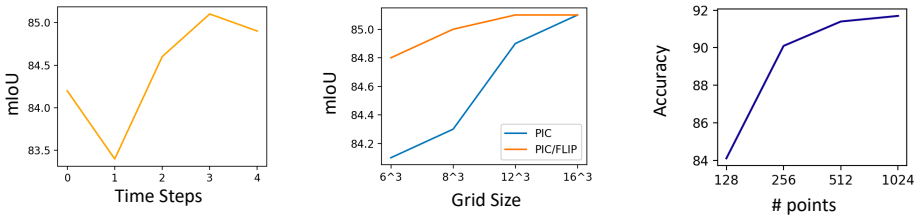

Figure 4: **Ablation Test**. Left) We test the effect of time steps on ShapeNet. Middle) We test the performance with different grid resolution on ShapeNet. We compare the networks using PIC and PIC/FLIP on the same figure. Right) We test the performance with number of points on ModelNet40.

the ShapeNet data set (Yi et al., 2016). The mIoU reached 85.1% with the advection module in comparison to 84.3% without it, necessitating the role of the advection step.

Next, We also tested the performance of advection with different number of time steps, evaluated mIoU on ShapeNet. We got the performance of 83.4%, 84.6%, 85.1% and 84.9% with one, two, three, and four time steps, indicating the necessity of the multiple-step advection. Meanwhile, these results imply the risk of over-fitting due to the multiple iterations and we suggest that the number of steps depends on the data set and the model.

We then performed test for a frozen velocity field over the entire learning iterations. We only trained one velocity field and reuse for multiple times for the whole loop. This change resulted in the decrease of the mIoU from 85.1% to 83.9% (is only slight better than our standard network with a single time step). This result endorses the importance of the temporally varying velocity.

Grid and data resolution We also evaluated the network performance with different grid resolutions. We tested the mIoU on ShapeNet. We get results of 84.8%, 85.0%, 85.1% and 85.1% with the resolution of 6^3, 8^3, 12^3 and 16^3, showing that our method can work with a low-resolution grid. We also tested our network's robustness against number of points on ModelNet40 data set. We get the performance of 91.7%, 91.4%, 90.1% and 84.1% with number of points of 1024, 512, 256 and 128, showing strong robustness against missing points.

PIC/FLIP interpolation We highlight the role of our momentum-conserving interpolation scheme by testing the different variations. In the segmentation task, we removed the normalization part in Equation 2 to accumulate the grid features from their neighbors without smoothing. This change results in the decrease of the mIoU from 85.1% to 79.3%. Also, to highlight the importance of the PIC/FLIP advection scheme, we compared the mIoU on ShapeNet with PIC/FLIP and with only PIC. Both methods reach 85.1% on a 16^3 grid. But the performance dropped to 84.1%, 84.3%, 84.9% with grid resolution of 6^3, 8^3, 12^3 with PIC only.

5.2 Applications

Classification We tested our network on ModelNet40 (Z. Wu, 2015) and ModelNet10 for classification. ModelNet40 contains 12,311 3D mesh models from 40 categories, among which 9843 models are training set and 2468 models are testing set. We used the point cloud data set (Qi et al., 2016a) with 2048 points uniformly sampled from these 3D mesh models. The first 1024 points were used for both training and testing. ModelNet10 is a subset of ModelNet40 with only ten different classes. For data augmentation, we randomly rotated objects along the up-axis in addition to randomly dropping and jittering points by $N(0, 0.02)$. We use a grid resolution 12^3 and 16^3 to train the ModelNet40 and ModelNet10 respectively. As shown in Table 1, our result is among the state-of-art with only minor parameter tuning.

Segmentation We tested our algorithm for object part segmentation on ShapeNet (Yi et al., 2016), which contains 16,881 mesh models from 16 categories. We used a grid resolution 16^3 for training and testing. We showed the state-of-art performance of our approach in Table 2. Notice that part of our result is close to DGCNN in complex shapes with a small portion in the data set, such as car,

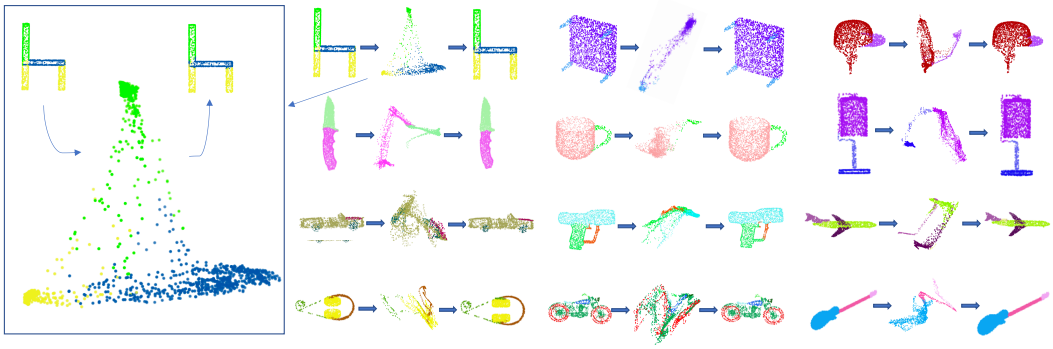

Figure 5: **Visualization of segmentation**. We visualize examples for most categories. Each example consists of initial shape, intermediary grouping, and final part prediction.

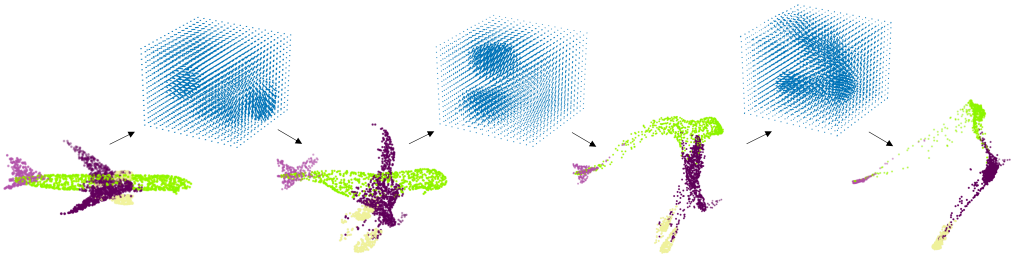

Figure 6: This figure visualize the advection of an air plane in each steps. The first is the origin point cloud and the following three are the three timestamps in the advection. We also visualize the velocity field for each timestamp. Note that we rotate the point cloud and normalize the velocity field for visualization purpose.

motor and rocket (see Figure 5). More examples animating the segmentation process can be seen in Figure 5. In Figure 6, we visualized the whole advection process for an airplane showing the point evolution. We also tested our algorithm for semantic segmentation in scenes on the Stanford Large-Scale 3D Indoor Spaces Data set (S3DIS) (Armeni et al., 2016). Notice that we only use a grid resolution 8^3 for this test case. As shown in Table 3, the performance of our model is comparable to the state-of-the-art for a large scale real-world data.

Table 1: Classification results on ModelNet10 and ModelNet40.

Method	Input	ModelNet10	ModelNet40
Points with only XYZ			
SO-Net	2048 points	94.1	90.9
PCNN	1024 points	94.9	92.3
PointNet	1024 points	-	89.2
PointGrid	1024 points	-	92.0
DGCNN	1024 points	-	92.9
PointCNN	1024 points	-	92.5
Ours ($16^3/12^3$)	1024 points	94.3	91.7
Additional input features			
O-CNN	octree (256^3) with normals	-	86.5
VoxNet	grid (32^3)	-	83.0
Kd-Net	32k points to kd-tree(depth 15)	94.0	91.8
FPNN	grid (64^3)	-	87.5
PointNet++	5000 points with normals	-	91.9
SpiderCNN	1024 points with normals	-	92.4
SO-Net	5000 points with normals	95.7	93.4
Ours ($16^3/12^3$)	1024 points with normals	95.7	93.0

Table 2: Segmentation results on ShapeNet.

Method	input	mIoU	aero	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skate board	table
Points with only XYZ																		
PointNet	2k pnts	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
PCNN	2k pnts	85.1	82.4	80.1	85.5	79.5	90.8	73.2	91.3	86.0	85.0	95.7	73.2	94.8	83.3	51.0	75.0	81.8
Kd-Net	4k pnts	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
DGCNN	2k pnts	85.1	84.2	83.7	84.4	77.1	90.9	78.5	91.5	87.3	82.9	96.0	67.0	93.3	82.6	59.7	75.5	82.0
PointCNN	2k pnts	86.1	84.1	86.4	86.0	80.8	90.6	79.7	92.3	88.4	85.3	96.1	77.2	95.2	84.2	64.2	80.0	82.9
Ours (16^3)	2k pnts	85.1	82.8	86.0	84.8	77.8	90.6	71.7	91.3	86.5	82.9	95.8	66.9	94.5	80.7	60.2	74.1	83.0
Additional input features																		
PointNet++	pnts, nors	85.1	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
SO-Net	pnts, nors	84.9	82.8	77.8	88.0	77.3	90.6	73.5	90.7	83.9	82.8	94.8	69.1	94.2	80.9	53.1	72.9	83.0
O-CNN + CRF	nors	85.9	85.5	87.1	84.7	77.0	91.1	85.1	91.9	87.4	83.3	95.4	56.9	96.2	81.6	53.5	74.1	84.4
SpiderCNN	pnts, nors	85.3	83.5	81.0	87.2	77.5	90.7	76.8	91.1	87.3	83.3	95.8	70.2	93.5	82.7	59.7	75.8	82.8
Ours (16^3)	pnts. nors	85.3	82.8	73.8	87.9	79.6	90.8	69.8	91.5	87.0	83.2	96.2	70.7	95.1	83.1	62.1	75.8	82.4
Train Separately																		
SPLATNet	pnts, img	85.4	83.2	84.3	89.1	80.3	90.7	75.5	92.1	87.1	83.9	96.3	75.6	95.8	83.8	64.0	75.5	81.8
Ours (16^3)	2k pnts	86.1	84.4	82.6	85.6	81.6	91.1	74.2	92.1	87.3	84.7	96.2	72.3	95.9	82.8	58.8	74.8	83.3

Table 3: Segmentation results on S3DIS.

Method	mIoU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
Cross validation result														
PointNet	47.6	88.0	88.7	69.3	42.4	23.1	47.5	51.6	54.1	42.0	9.6	38.2	29.4	35.2
SPGraph	62.1	89.9	95.1	76.4	62.8	47.1	55.3	68.4	73.5	69.2	63.2	45.9	8.7	52.9
PointCNN	65.4	94.8	97.3	75.8	63.3	51.7	58.4	57.2	71.6	69.1	39.1	61.2	52.2	58.6
Ours (8^3)	60.8	93.0	95.3	76.5	46.4	43.6	62.1	62.2	68.0	65.1	30.1	48.0	49.6	50.3
Area 5 result														
PointNet	41.09	88.80	97.33	69.80	0.05	3.92	46.26	10.76	58.93	52.61	5.85	40.28	26.38	33.22
SPGraph	58.04	89.35	96.87	78.12	0.00	42.81	48.93	61.58	84.66	75.41	69.84	52.60	2.10	52.22
SegCloud	48.92	90.06	96.05	69.86	0.00	18.37	38.35	23.12	70.40	75.89	40.88	58.42	12.96	41.60
PCCN	58.27	92.26	96.20	75.89	0.27	5.98	69.49	63.45	66.87	65.63	47.28	68.91	59.10	46.22
PointCNN	57.26	92.31	98.24	79.41	0.00	17.60	22.77	62.09	74.39	80.59	31.67	66.67	62.05	56.74
Ours (8^3)	56.65	94.07	97.91	79.14	0.16	15.42	43.62	62.68	79.35	75.18	29.51	57.02	59.12	43.23

6 Discussion and Conclusion

This paper presents a new perspective in treating the point cloud learning problem as a dynamic advection problem in a learned background velocity field. The learning mechanism and the evolution process is inspired by the fluid mechanics principles and the network architecture design is motivated by the classic PIC/FLIP numerical scheme. The key technical contribution of the proposed approach is to jointly define the point cloud learning problem as a flow advection problem in a world space using a static background grid and the local space using moving particles.

Compared with the previous hybrid grid-point learning methods, e.g. two-way coupled particle-grid schemes (Su et al., 2018; Atzmon et al., 2018; Liu et al., 2019), our approach solves the learning problem from a dynamic system perspective which accumulates features in a flow field learned temporally. The coupled Eulerian-Lagrangian data structure in conjunction with its accommodated G2P and P2G interpolation schemes provide a complete solution to tackle the computational challenges regarding stencil construction and feature evolution by leveraging a numerical infrastructure that is matured in the scientific computing community. On another hand, our approach can be thought of as a preliminary exploration in creating a new physical reservoir motivated by continuum mechanics in order to find alternative solutions for the conventional point cloud processing networks. Such reservoir computing techniques exhibit strong physical intuitions in the network mechanism design and provide a intrinsically continuous interface for the feature evolution.

The main limitation of the proposed approach is its capability in scaling to a grid with higher resolution, which has been the main focus of developing a high-performance scientific computing algorithm for physical simulation. Our future plan is to scale the algorithm to handle more complex point clouds with sparse and adaptive grid structures to take advantage of the computational power of modern platforms. We also plan to explore more geometric reservoirs and temporal evolution schemes from continuous physical simulations in order to accommodate the various emerging learning problems that exhibit characteristics of natural systems.

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J. Guibas. Representation learning and adversarial generation of 3d point clouds. *CoRR*, abs/1707.02392, 2017.

Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *CoRR*, abs/1803.10091, 2018.

J. U. Brackbill, D. B. Kothe, and H. M. Ruppel (eds.). *FLIP (Fluid-Implicit-Particle): A low-dissipation, particle-in-cell method for fluid flow*, April 1987.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *CoRR*, abs/1611.08097, 2016.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. *CoRR*, abs/1801.10130, 2018.

Weinan E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, Mar 2017. ISSN 2194-671X. doi: 10.1007/s40304-017-0103-z.

M.W. Evans and F.H. Harlow. The particle-in-cell method for hydrodynamic calculations. 6 1957.

Haoqiang Fan, Hao Su, and Leonidas J. Guibas. A point set generation network for 3d object reconstruction from a single image. *CoRR*, abs/1612.00603, 2016.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pp. 15–22, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383260.

Yifan Feng, Zizhao Zhang, Xibin Zhao, Rongrong Ji, and Yue Gao. Gvcnn: Group-view convolutional neural networks for 3d shape recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pp. 23–30, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383261. URL `http://doi.acm.org/10.1145/383259.383261`.

Helmut Hauser, Auke J. Ijspeert, Rudolf M. Füchslin, Rolf Pfeifer, and Wolfgang Maass. The role of feedback in morphological computation with compliant bodies. *Biological Cybernetics*, 106 (10):595–613, Nov 2012. ISSN 1432-0770. doi: 10.1007/s00422-012-0516-4.

Herbert Jaeger. The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148, 01 2001.

Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In *NIPS*, 2002.

Azarakhsh Jalalvand, Glenn Wallendael, and Rik Van de Walle. Real-time reservoir computing network-based systems for detection tasks on visual contents. pp. 146–151, 06 2015. doi: 10.1109/CICSyN.2015.35.

Mingyang Jiang, Yiran Wu, and Cewu Lu. Pointsift: A sift-like network module for 3d point cloud semantic segmentation. *CoRR*, abs/1807.00652, 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

Roman Klokov and Victor S. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. *CoRR*, abs/1704.01222, 2017.

Truc Le and Ye Duan. Pointgrid: A deep network for 3d shape understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Yann Lecun, Leon Bottou, Y Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998. doi: 10.1109/5.726791.

Jiaxin Li, Ben M. Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. *CoRR*, abs/1803.04249, 2018a.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 820–830. Curran Associates, Inc., 2018b.

Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel CNN for efficient 3d deep learning. *CoRR*, abs/1907.03739, 2019.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-net: Learning PDEs from data. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3208–3216, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Mantas Lukoeviius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127 – 149, 2009. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2009.03.005.

Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.*, 14 (11):2531–2560, November 2002. ISSN 0899-7667. doi: 10.1162/089976602760407955.

D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *IROS*, 2015.

Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30(1):543–574, 1992.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016a.

Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5648–5656, 2016b.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. *CoRR*, abs/1611.05009, 2016.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *CoRR*, abs/1804.04272, 2018.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. *CoRR*, abs/1802.08275, 2018.

Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Hroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100 – 123, 2019. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2019.03.005.

Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. *CoRR*, abs/1710.07563, 2017.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4), 2017.

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

Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. *CoRR*, abs/1803.11527, 2018.

Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Interpretable unsupervised learning on 3d point clouds. *CoRR*, abs/1712.07262, 2017.

Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016.

Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. *CoRR*, abs/1801.06761, 2018.

A. Khosla F. Yu L. Zhang X. Tang J. Xiao Z. Wu, S. Song. 3d shapenets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition*, 2015.