Learned Gridification for Efficient Point Cloud Processing

Putri A. van der Linden[*1] **David W. Romero**[*2] **Erik J. Bekkers**[1]

Abstract

Neural operations that rely on neighborhood information are much more expensive when deployed on point clouds than on grid data due to the irregular distances between points in a point cloud. In a grid, on the other hand, we can compute the kernel only once and reuse it for all query positions. As a result, operations that rely on neighborhood information scale much worse for point clouds than for grid data, specially for large inputs and large neighborhoods.

In this work, we address the scalability issue of point cloud methods by tackling its root cause: the irregularity of the data. We propose *learnable gridification* as the first step in a point cloud processing pipeline to transform the point cloud into a compact, regular grid. Thanks to gridification, subsequent layers can use operations defined on regular grids, e.g., `Conv3D`, which scale much better than native point cloud methods. We then extend gridification to point cloud to point cloud tasks, e.g., segmentation, by adding a *learnable de-gridification* step at the end of the point cloud processing pipeline to map the compact, regular grid back to its original point cloud form. Through theoretical and empirical analysis, we show that *gridified networks* scale better in terms of memory and time than networks directly applied on raw point cloud data, while being able to achieve competitive results. Code is available at `https://github.com/computri/gridifier`.

1. Introduction

Point clouds provide sparse geometric representations of objects or surfaces equipped with signals defined over their structure, e.g., the surface normals of an underlying object (Wu et al., 2015; Qi et al., 2017a) or the chemical properties of a molecule (Ramakrishnan et al., 2014; Schütt et al., 2017). Several neural operators have been developed that can be applied to such sparse representations provided by point clouds. These methods can be broadly understood as continuous generalizations of neural operators originally defined over regular discrete grids, e.g., convolution (Wu et al., 2019) and self-attention (Zhao et al., 2021).

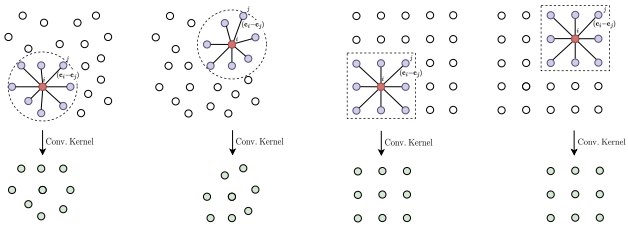

Figure 1. Convolution on point clouds and grids. Due to the irregular nature of point clouds, convolutional kernels –and other operations based on neighborhood information– must be re-rendered for every query point in the point cloud (left). In contrast, grid data is regularly arranged, and thus pairwise distances are equal for any query point in the grid (right). As a result, the convolutional kernel can be computed once and reused for all query points.

The problem of learning on raw point clouds. Unfortunately, the flexibility required from neural operators to accommodate irregular sparse representations like point clouds brings about important increases in time and memory consumption. This is especially prominent in neural operations that construct feature representations based on neighborhood information, e.g., convolution. In the case of point clouds, the irregular distances between points make these neural operations significantly more computationally demanding compared to regular grid representations like images or text. For instance, for convolution, the convolutional kernel needs to be recalculated for each point in a point cloud to account for irregular distances from the query point to other points in its neighborhood (Fig. 1 left). In contrast, grid representations standardize pairwise distances following a grid structure (Fig. 1 right). As a result, the distances from a point to all other points in its neighborhood are fixed for all points queried in the grid. Therefore, it is possible to compute the kernel once, and reuse it across all query positions. This difference illustrates that operations relying on neighborhood information scale much worse in terms of memory and time for point clouds than for grid

[*]Equal contribution. [1]University of Amsterdam. [2]Vrije Universiteit Amsterdam, The Netherlands. Correspondence to: Putri A. van der Linden <p.a.vanderlinden@uva.nl>, David W. Romero <d.w.romeroguzman@vu.nl>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

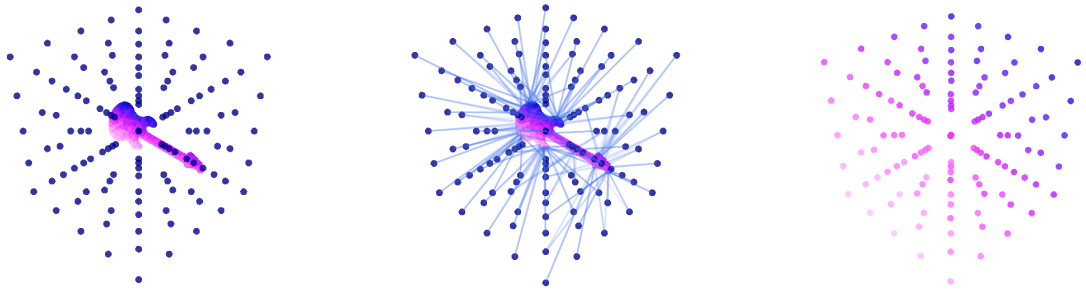

Figure 2. Gridification. Gridification maps a point cloud $\mathcal{P}$ onto a compact regular grid $\mathcal{G}$. The method first constructs a D-dimensional grid (left) that overlaps the point cloud. Then, it connects points on the point cloud to points in the grid given by a connectivity scheme $\mathcal{E}_{\mathcal{P}\to\mathcal{G}}$, i.e., a set of edges from points in the point cloud to points in the grid, determined by *bilateral k-nearest neighbors connectivity* (middle). Finally, gridification propagates information from the point cloud onto the grid through a *convolutional message passing* layer acting over the bipartite graph $(\mathcal{P}, \mathcal{G}, \mathcal{E}_{\mathcal{P}\to\mathcal{G}})$. By carefully selecting the different components of the gridification module, gridification is able to construct compact rich grid representations that can be subsequently processed with grid operations such as `Conv3D`.

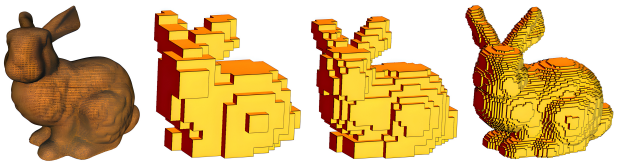

Figure 3. Voxelization of the Stanford Bunny (Turk & Levoy, 1994) for different resolutions. Taken from Karmakar et al. (2011).

data, specially for large inputs and large neighborhoods.

A potential solution: Voxelization. A potential solution to address the challenges posed by point clouds lies in treating the point cloud as a continuous density that can be sampled on a dense regular grid: a process called *voxelization* (Maturana & Scherer, 2015; Wu et al., 2015). The idea of voxelization is to create a grid that overlaps with the domain of the point cloud (Fig. 3). Although voxelization methods create grid representations on which neural operations defined on grids can act, e.g., `Conv3D`, the grids resulting from voxelization are oftentimes much larger than the number of points in the original point cloud. This is a consequence of (*i*) the high-resolution grids required to describe fine details from the point cloud, and *(ii)* the low occupancy of the grid resulting from the sparse nature of point clouds which generally leads to many more points to process in the resulting grid than in the original point cloud.

Our proposed solution: Gridification. In this paper, we propose an alternative solution to address the memory and computational scalability of point cloud methods by addressing its root cause: *the irregularity of the data*. We propose *learnable gridification* as the first step in a point cloud processing pipeline to transform the point cloud into a *compact, regular grid* (Fig. 2). Thanks to gridification, subsequent layers can use operations defined on grids, e.g., `Conv3D`, which scale much better than native point cloud methods. In a nutshell, gridification can be understood as a *convolutional message passing* layer acting on a *bipartite graph* that establishes connections between points in the point cloud to points in the grid given by a *bilateral k-nearest neighbor connectivity*. The proposed bilateral k-nearest neighbor connectivity guarantees that all points both in the point cloud and in the grid are connected, therefore allowing for the construction of expressive yet compact grid representations.

In contrast to voxelization, gridification produces expressive compact grid representations in which the number of points in the resulting compact regular grid is roughly equal to the number of points in the original point cloud, yet the grid is able to preserve fine geometric details from the original point cloud. For instance, we observe that point clouds with N=1000 points can be effectively mapped to a compact dense 10x10x10 grid without significant information loss. We show through theoretical and empirical analysis that the resulting grid representations scale much better in terms of memory and time than native point cloud methods. This is verified on several comparison studies for increasing number of points in the point cloud and increasing neighborhood sizes in the construction of convolutional kernels.

We demonstrate that gridification can also be used for tasks from point clouds to point clouds, e.g., segmentation. To this end, we introduce a *learnable de-gridification* step at the end of the point cloud processing pipeline, which can be seen as an inverted gridification step that maps the compact, regular grid back to its original point cloud form. This extension allows for the construction of *gridified networks* –networks that operate on grids– to solve global prediction tasks, e.g., classification, as well as dense prediction tasks, e.g., segmentation and regression, on point cloud data.

2. Method

2.1. Point cloud and grid representations

Point cloud. A point cloud $\mathcal{P}=\{(\mathbf{c}_i^{\mathcal{P}}, \mathbf{x}_i^{\mathcal{P}})\}_{i=1}^{\mathrm{N}_{\mathcal{P}}}$ is an *unstructured* set of $\mathrm{N}_{\mathcal{P}}$ pairs of coordinate-feature values $(\mathbf{c}_i^{\mathcal{P}}, \mathbf{x}_i^{\mathcal{P}})$ scattered in space without any predefined pattern or connectivity. Point clouds sparsely represent geometric structures

through pairs of coordinate vectors $\mathbf{c}_i^{\mathcal{P}} \in \mathbb{R}^{\mathrm{D}}$ and corresponding function values over that geometric structure $\mathbf{x}_i^{\mathcal{P}} \in \mathbb{R}^{\mathrm{F}_{\mathcal{P}}}$, e.g., surface normals, RGB-values, electric potentials, etc.

Grid. A grid $\mathcal{G}=\{(\mathbf{c}_i^{\mathcal{G}}, \mathbf{x}_i^{\mathcal{G}})\}_{i=1}^{\mathrm{N}_{\mathcal{G}}}$ can be interpreted as a point cloud on which the coordinate-feature pairs $(\mathbf{c}_i^{\mathcal{G}}, \mathbf{x}_i^{\mathcal{G}})$ are arranged in a regular pattern that form a lattice. In contrast to general point clouds, points in a grid are evenly spaced and align along predefined axes, e.g., x, y, z. The regular spacing between points leads to regular pairwise distances for all query points in the grid. As a result, we can calculate pairwise attributes once, and reuse them for all query points.

2.2. Gridification: From a point cloud to a dense grid

We seek to map the sparse point cloud $\mathcal{P}=\{(\mathbf{c}_j^{\mathcal{P}}, \mathbf{x}_j^{\mathcal{P}})\}_{j=1}^{\mathrm{N}_{\mathcal{P}}}$ onto a compact regular grid $\mathcal{G}=\{(\mathbf{c}_i^{\mathcal{G}}, \mathbf{x}_i^{\mathcal{G}})\}_{i=1}^{\mathrm{N}_{\mathcal{G}}}$ in $\mathbb{R}^{\mathrm{D}}$. We formalize this process as an operation over a *bipartite graph* that establishes connections between points in the point cloud $\mathcal{P}$ to points in the grid $\mathcal{G}$ given by a *connectivity scheme* $\mathcal{E}_{\mathcal{P}\to\mathcal{G}}$ defined as a set of edges $\mathbf{e}_{j\to i} \in \mathcal{E}_{\mathcal{P}\to\mathcal{G}}$.

Learnable gridification as message passing. We aim to learn a mapping from $\mathcal{P}$ to $\mathcal{G}$ such that the grid representation $\mathcal{G}=\{(\mathbf{c}_i^{\mathcal{G}}, \mathbf{x}_i^{\mathcal{G}})\}_{i=1}^{\mathrm{N}_{\mathcal{G}}}$, $\mathbf{c}_i \in \mathbb{R}^{\mathrm{D}}$, $\mathbf{x}_i^{\mathcal{G}} \in \mathbb{R}^{\mathrm{F}_{\mathcal{G}}}$, adequately represents the source point cloud $\mathcal{P}$ for the downstream task. Given a source point cloud $\mathcal{P}$, a target grid $\mathcal{G}$ and a connectivity scheme $\mathcal{E}_{\mathcal{P}\to\mathcal{G}}$, we define gridification as a *convolutional message passing* layer (Gilmer et al., 2017) on the bipartite graph $(\mathcal{P}, \mathcal{G}, \mathcal{E}_{\mathcal{P}\to\mathcal{G}})$ defined as:

$$\mathbf{x}_i^{\mathcal{G}} = \phi_{\mathrm{upd}}\left(\bigoplus_{\mathbf{e}_{j\to i} \in \mathcal{E}_{\mathcal{P}\to\mathcal{G}}} \phi_{\mathrm{msg}}\Big(\phi_{\mathrm{node}}\left(\mathbf{x}_j^{\mathcal{P}}\right), \phi_{\mathrm{pos}}\left(\mathbf{c}_i^{\mathcal{G}} - \mathbf{c}_j^{\mathcal{P}}\right)\Big)\right). \tag{1}$$

It consists of a node embedding network $\phi_{\mathrm{node}} : \mathbb{R}^{\mathrm{F}_{\mathcal{P}}} \to \mathbb{R}^{\mathrm{H}}$ that processes the point cloud features $\mathbf{x}_i^{\mathcal{G}}$, a positional embedding network $\phi_{\mathrm{pos}} : \mathbb{R}^{\mathrm{D}} \to \mathbb{R}^{\mathrm{H}}$ that creates feature representations based on the pairwise distances between coordinates in $\mathcal{G}$ and $\mathcal{P}$ –thus resembling a convolutional kernel–, a message embedding network $\phi_{\mathrm{msg}} : \mathbb{R}^{2\mathrm{H}} \to \mathbb{R}^{\mathrm{H}}$ that receives both the node embedding and the relative position embedding to create the so-called *message*. After the messages are created for all nodes described by connectivity of the node, these features are aggregated via the aggregation function $\bigoplus$, e.g., $\max$, mean. Finally, the aggregated message is passed through the update network $\phi_{\mathrm{upd}} : \mathbb{R}^{\mathrm{H}} \to \mathbb{R}^{\mathrm{F}_{\mathcal{G}}}$ to produce the grid feature representations $\mathbf{x}_i^{\mathcal{G}} \in \mathbb{R}^{\mathrm{F}_{\mathcal{P}}}$.

2.3. De-gridification: From a dense grid to a point cloud

To extend the use of gridification to tasks from the point cloud $\mathcal{P}$ to the point cloud $\mathcal{P}$, e.g., segmentation, regression, we define a *de-gridification* step that sends a grid representation $\mathcal{G}$ back to its original point cloud form $\mathcal{P}$. Formally, the de-gridification step is defined as:

$$\mathbf{x}_i^{\mathcal{P}} = \phi_{\mathrm{upd}}\left(\bigoplus_{\mathbf{e}_{j\to i} \in \mathcal{E}_{\mathcal{G}\to\mathcal{P}}} \phi_{\mathrm{msg}}\Big(\phi_{\mathrm{node}}\left(\mathbf{x}_j^{\mathcal{G}}\right), \phi_{\mathrm{pos}}\left(\mathbf{c}_i^{\mathcal{P}} - \mathbf{c}_j^{\mathcal{G}}\right)\Big)\right). \tag{2}$$

Intuitively, de-gridification can be interpreted as a gridification step from $\mathcal{G}$ to $\mathcal{P}$ given by an inverted connectivity scheme $\mathcal{E}_{\mathcal{G}\to\mathcal{P}}=(\mathcal{E}_{\mathcal{P}\to\mathcal{G}})^{-1}$. Note that, it is not necessary to calculate the connectivity scheme for the de-gridification step. Instead, we can obtain it simply by taking the connectivity scheme from the gridification step $\mathcal{E}_{\mathcal{G}\to\mathcal{P}}$ and inverting the output and input nodes of the edges.

2.4. Requirements and properties of gridification

We desire to construct a compute and memory efficient grid representation $\mathcal{G}$ that captures all aspects of the point cloud $\mathcal{P}$ as good as possible. That is, a compact, yet rich grid representation $\mathcal{G}$ that preserves the structure of the point cloud $\mathcal{P}$ with as low loss of information as possible. With this goal in mind, we identify the following requirements:

(*i*) The number of points in the grid $\mathrm{N}_{\mathcal{G}}$ should be at least as large as the number of points in the point cloud $\mathrm{N}_{\mathcal{P}}$.

(*ii*) The width of all hidden representations of the node embedding network ϕ_{node} should be *at least as large* as the width of the point cloud features $\mathbf{x}_i^{\mathcal{P}}$, i.e., $\mathrm{F}_{\mathcal{P}}$.

(*iii*) The width of all hidden representations of the position embedding network ϕ_{pos} should be at least as large as the dimension of the domain D.

(*iv*) The width of all hidden representations of the embedding networks ϕ_{upd}, ϕ_{msg} should be *at least as large* as the width of the point cloud features $\mathbf{x}_i^{\mathcal{P}}$ plus the dimension of the domain D.

(*v*) Each point $\mathbf{c}^{\mathcal{P}}$ in the point cloud should be connected to *at least* one point $\mathbf{c}^{\mathcal{G}}$ in the grid.

(*vi*) The positional embedding network ϕ_{pos} should be able to describe high frequencies.

(*vii*) Each point $\mathbf{c}^{\mathcal{G}}$ in the grid should be connected to *at least* one point $\mathbf{c}^{\mathcal{P}}$ in the point cloud.

Preventing information loss. To prevent information loss, we want to avoid any kind of compression either in the grid representation or in any intermediary representation during the gridification process. Consequently, we restrict the number of points as well as the width of all representations to be at least as big as the corresponding dimensions in the source point cloud $\mathcal{P}$ –items (*i*)-(*iv*)–. In addition, we must make sure that all points in the point cloud are connected to points in the grid to prevent points from being disregarded during gridification –item (*v*)–. Finally, we must also make sure that the positional embedding network ϕ_{pos} is able to represent high frequencies –item (*vi*)–. This is important as multilayer perceptrons (MLPs) with piecewise nonlinearities, e.g., ReLU, have been shown to have an implicit bias

towards smooth functions (Tancik et al., 2020; Sitzmann et al., 2020). In the context of gridification, this means that using conventional MLPs for the positional embedding network ϕ_{pos} could result in over-smooth grid representations unable to represent fine details from the source point cloud. We circumvent this issue by using parameterizations for ϕ_{pos} able to model high frequencies (Sec. 2.5.3).

Encouraging compact representations. In addition to encouraging no information loss, we also identify requirements that encourage the resulting grid representation to be compact and expressive. First, we note that item (*v*) is important for this end as well, as over-smooth representations implicitly require higher resolutions to be able to encode fine-grained details. Additionally, we impose all points in the grid to be connected to points in the point cloud –item (*vii*)– to prevent the grid representation from having low occupancy. This restriction allows us to make sure that all the spatial capacity of the grid is being used. This in turn allows us to construct compact rich grid representations.

2.5. Materializing the gridification module

Based on the previous requirements and properties, we define the components of the gridification module as follows:

2.5.1. The grid $\mathcal{G}$

Let $[a,b]^{\text{D}}$ be the domain of the point cloud $\mathcal{P}$, i.e., $\mathbf{c}_i^{\mathcal{P}} \in [a,b]^{\text{D}}$, $\forall\ \mathbf{c}_i^{\mathcal{P}} \in \mathcal{P}$. Then, we define the regular grid $\mathcal{G}$ over the same domain $[a,b]^{\text{D}}$ with $\sqrt[\text{D}]{\text{N}^{\mathcal{G}}}$ points along each dimension. By doing so, we guarantee that the grid $\mathcal{G}$ is uniformly spaced over the domain of the point cloud, therefore (*i*) preserving the statistics of the input point cloud, and (*ii*) being able to represent the underlying signal in the same range. In practice, point clouds are normalized during the preprocessing steps preceding a point cloud processing pipeline. As a result, we often have that a= − 1 and b=1, leading to a point cloud and a grid defined on $[-1,1]^{\text{D}}$.

2.5.2. The connectivity scheme $\mathcal{E}_{\mathcal{P}\to\mathcal{G}}$

Motivated by the requirements in Sec. 2.4, we opt for *bilateral k-nearest neighbor connectivity* over common alternatives such as radius connectivity (Qi et al., 2017a;b) or one-way k-nearest neighbor connectivity (Barber et al., 1996; Connor & Kumar, 2010) for the construction of the connectivity scheme $\mathcal{E}_{\mathcal{P}\to\mathcal{G}}$ to guarantee that no points either in the grid $\mathcal{G}$ nor the point cloud $\mathcal{P}$ are disconnected. Bilateral k-nearest neighbor connectivity consists of a two-way k-nearest neighbor approach in which first each point $\mathbf{c}_i^{\mathcal{G}}$ in the grid is linked to the k nearest points $\mathbf{c}_j^{\mathcal{P}}$ in the point-cloud. Subsequently, connections are established from each point $\mathbf{c}_i^{\mathcal{P}}$ in the point cloud to its nearest k points $\mathbf{c}_j^{\mathcal{G}}$ in the grid (Fig. 4). By following this procedure, bilateral k-nearest neighbor connectivity creates a *complete* connectivity scheme, i.e., with no disconnected points, from $\mathcal{P}$ to $\mathcal{G}$ with at least k and at most $2k$ connections for each point.

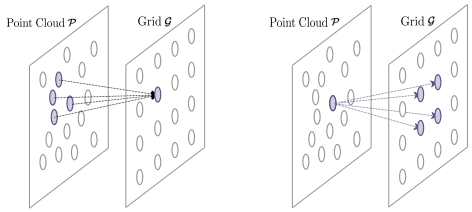

Figure 4. Bilateral k-nearest neighbor connectivity for k=4.

2.5.3. The positional embedding network ϕ_{pos}

In literature, the positional embedding network ϕ_{pos} is often parameterized as an MLP with piecewise nonlinearities, e.g., ReLU, that receives relative positions $(\mathbf{c}_i - \mathbf{c}_j)$ as input and retrieves the value of an spatial function at that position $\phi_{\text{pos}}(\mathbf{c}_i - \mathbf{c}_j)$ (Schütt et al., 2017; Qi et al., 2017b; Wu et al., 2019). However, previous studies have shown that MLPs with piecewise nonlinearities suffer from an spectral bias towards low frequencies, which limits their ability to represent functions with high frequencies (Tancik et al., 2020; Sitzmann et al., 2020). In the context of modelling spatial neural operators such as ϕ_{pos}, this implies that using piecewise MLPs to parameterize spatial neural operators leads to inherently smooth operators. Consequently, applying such an operator over an input function, e.g., via a convolution operation, would implicitly perform a low-pass filtering of the input, causing the output representations to lack information regarding fine-grained details of the input.

To overcome this issue, we rely on the insights from *Continuous Kernel Convolutions* (Romero et al., 2021) and parameterize the positional embedding network as a *Neural Field* (Sitzmann et al., 2020; Tancik et al., 2020). In contrast to piecewise MLPs, neural fields easily model high frequencies, and thus allow for powerful parameterizations of spatial neural operators that do not perform smoothing. In the context of gridification, using neural fields to parameterize ϕ_{pos} allows gridification to project fine-grained geometric information from the point cloud onto the grid.

2.6. Gridified networks for global and dense prediction

Gridification and de-gridification allow for the construction of *gridified networks* able to process point clouds both for global and dense prediction tasks (Fig. 5). For global prediction tasks, e.g., classification, we construct a point cloud processing pipeline consisting of gridification, followed by a *grid network*, i.e., a neural network that operates on grid data, designed for global prediction, e.g., a ResNet (He et al., 2016) or a ViT (Dosovitskiy et al., 2020). For dense prediction tasks, e.g., segmentation, our proposed point cloud pipeline consists of gridification, followed by a grid network designed for dense predictions, e.g., a U-Net (Ronneberger et al., 2015) or a CCNN (Knigge et al., 2023). After the processed grid representation is obtained, we utilize the de-gridification step to map back the grid representation to a point cloud with the output node predictions.

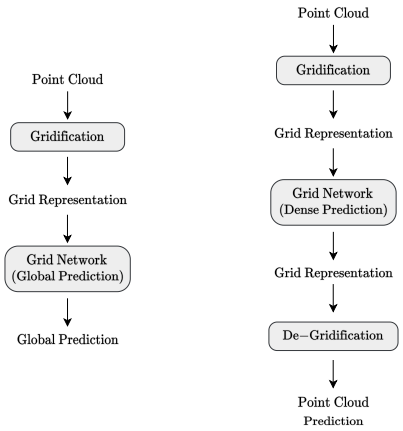

Figure 5. Point cloud processing pipeline for global prediction (left) and dense prediction tasks (right).

3. Related Work

Deep learning approaches for point cloud processing can be broadly classified in two main categories: (*i*) native point cloud methods and (*ii*) voxelization methods.

Native point cloud methods. Native point cloud methods operate directly on the raw, irregular point cloud data without any preprocessing steps such as voxelization. These methods leverage the inherent spatial distribution of the points to extract meaningful features. PointNet (Qi et al., 2017a) introduced a pioneering framework for point cloud processing by employing shared multilayer perceptrons and symmetric functions to learn global and local features from unordered point sets. PointNet++ (Qi et al., 2017b) extended this work with hierarchical neural networks to capture hierarchical structures in point clouds. PointConv (Wu et al., 2019) introduced a convolution operation specifically designed for point clouds, incorporating local coordinate systems to capture local geometric structures. PointGNN (Shi & Rajkumar, 2020) utilized graph neural networks to model interactions between neighboring points in point clouds.

Despite the flexibility in handling irregular data that native point cloud methods provide, they suffer from scalability issues due to the increased computational and memory complexity of processing unstructured point sets.

Voxelization methods. Voxelization methods aim to convert the irregular point cloud data into a regular grid structure, enabling the utilization of neural architectures designed for regular grid data. VoxNet (Maturana & Scherer, 2015) introduced the concept of voxelization for point clouds and employed 3D convolutions on the resulting grid representations. Volumetric CNN (Qi et al., 2016) extended this approach with an occupancy grid representation and achieved impressive performance on 3D shape classification tasks. Other works, such as VoxSegNet (Wang & Lu, 2019) explore variations of voxelization techniques to improve performance on tasks like object detection and segmentation.

While voxelization methods offer a well-founded solution to the computational and memory complexity of native point cloud methods, in practice, they suffer from high memory consumption and information loss due to the discretization process. This is due to the inherent trade-off between the need to capture fine geometric details –which requires high resolution grids–, and the need for efficiency –which favors low resolution grids–. As a result, conventional voxelization methods struggle to strike a balance between resolution and speed. In contrast, gridification is able to generate compact yet expressive grid representations able to preserve fine geometric details on a low resolution grid with roughly the same number of points as the source point cloud.

Hybrid methods. Aside from pure point cloud and voxelization methods, there exist works that attempt combine the advantages of both categories. Their main idea is to combine point-wise and grid-wise operations to perform effective feature extraction while maintaining scalability and efficiency. PointGrid (Le & Duan, 2018) uses a hybrid representation by voxelizing the point cloud and employing a combination of point-wise and grid-wise operations at each layer. Point-Voxel CNN (Liu et al., 2019) combines grid convolutions with point-wise feature extraction. It uses low-resolution voxelization to aggregate neighborhoods with regular 3D convolutions and MLPs to generate point-wise features that preserve fine-grained structure. These features are then fused through interpolation. Point-Voxel Transformer (Zhang et al., 2022) follows a similar two-branch structure, but replaces convolutions with windowed self-attention.

Although hybrid methods reduce the computational and memory complexity of native point cloud methods, their explicit use of voxelization still leads to a trade-off between information loss and efficiency on that branch. To compensate for the information lost during voxelization, they require a parallel raw point cloud branch, which does not scale well. In contrast, gridification does not make use of raw point cloud branches but instead focuses on the creation of descriptive compact grid representations that preserve the geometric information of the source point cloud. Hence, gridification offers a solution with better scalability properties than existing hybrid methods.

4. Experiments

To evaluate our approach, we first analyze the expressive capacity of gridification and de-gridification on a toy point cloud reconstruction task. Next, we construct gridified networks and evaluate them on classification and segmentation tasks. In addition, we provide empirical analyses on the computational and memory complexity of gridified networks which we then corroborate with theoretical analyses.

Experimental setup. For the position embedding function ϕ_{pos} we use an Random Fourier Feature Network (Tancik

Table 1. Classification performance on ModelNet40 benchmark.

MODEL	INPUT	TYPE	ACCURACY	PARAMETERS
PointNet++ (Qi et al., 2017b)	32×1000	native	89.64	1.5M
VoxNet (Maturana & Scherer, 2015)	32×30^3	voxelization	83.00	0.92M
PointGrid (Le & Duan, 2018)	32×16^3	voxelization	92.00	-
Point Voxel Transformer (Zhang et al., 2022)	32×1024	hybrid	94.00	2.76M
Gridified Networks 3x3x3 (Ours)	$32 \times 1000 \rightarrow 32 \times 3^3$	voxelization	90.86	0.28M
Gridified Networks 9x9x9 (Ours)	$32 \times 1000 \rightarrow 32 \times 9^3$	voxelization	92.28	0.47M

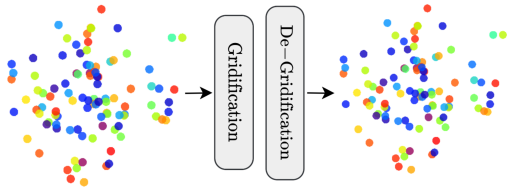

Figure 6. Random point clouds with random scalar node features are mapped to a grid representation. From the grid representation the node features need to be reconstructed via de-gridification.

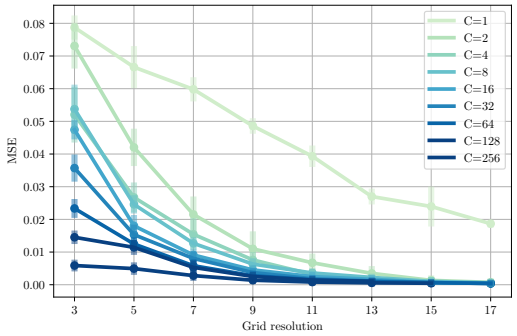

Figure 7. Random point cloud reconstruction error for varying grid resolution and number of channels on the grid representation.

et al., 2020), due to explicit control over the smoothness through the initial frequency parameter Ω. The practical setup and instantiation of the convolution blocks can be found in Appendix A. We train our models without data augmentation using AdamW (Loshchilov & Hutter, 2019) and a cosine scheduler (Loshchilov & Hutter, 2017) with 10 epochs of linear warm-up. We follow the standard procedure and preprocess all objects in the datasets to be centered and normalized. For each dataset, we choose the grid resolution such that its number of points is roughly equal to the size of the original point cloud. For ModelNet40 we use surface normals in addition to positions as node features. Dataset specific hyperparameters can be found in Appendix B.

4.1. Random point cloud reconstruction

First, we evaluate the expressivity of our proposed gridification and de-gridification procedure. To this end, we construct a dataset with 1000 synthetic random graphs –800 for training and 200 for validation– consisting of a predefined number of nodes $\mathrm{N}^{\mathcal{P}}$=1000 randomly sampled on the unit cube, i.e., $\mathbf{c}_i^{\mathcal{P}} \sim \mathcal{U}([-1,1]^3)$, accompanied with a random scalar feature $f_i^{\mathcal{P}} \sim \mathcal{U}(-1,1)$ at each position.

Experimental setup. To evaluate the expressiveness of our method, we set up a network consisting only of a gridification and a de-gridification step, i.e., no intermediary layers, in a point cloud reconstruction pipeline. In other words, the task consists of propagating the point cloud into a grid representation, and mapping the grid representation back to the original point cloud (see Fig. 6). Therefore, to successfully reconstruct the original point cloud from the grid representation, the grid representation must be able to retain sufficient information from the input point cloud.

Results. Fig. 7 shows reconstruction errors for different resolutions and different number of channels in the intermediary grid representation. We observe that it is possible to obtain good reconstructions by increasing the resolution of the grid or its number of channels. From an efficiency perspective, it is preferred to utilize low resolution representations with a larger number of channels due to the exponential growth in computational demands associated with higher grid resolutions, which instead scale linearly with the number of channels of the representation. Our experiments show that gridification is able to obtain compact grid representations that preserve the structure of the input point cloud. Furthermore, the quality of the grid representations can be efficiently improved by scaling the number of channels used.

4.2. ModelNet40 classification

Next, we evaluate gridification on point cloud classification. We deploy gridified networks on ModelNet40 (Wu et al., 2015): a synthetic dataset for 3D shape classification, consisting of 12,311 3D meshes of objects belonging to 40 classes. ModelNet40 is broadly used as a point cloud benchmark in which points are uniformly sampled from the faces of the meshes.

Results. Our results (Tab. 1) show that gridified networks achieve competitive performance while being significantly more efficient in terms of parameters, compute and memory. Interestingly, and in contrast to voxelization methods, we observe that gridified networks operate well even on extremely low resolution grids. For instance, on a 3×3×3 grid, gridified networks attain an accuracy of 90.86%.

4.3. ShapeNet part segmentation

Next, we evaluate gridification on point cloud segmentation. To this end, we deploy gridified networks on ShapeNet (Yi

Table 2. Segmentation performance on ShapeNet-part benchmark.

MODEL	Gridified Networks	PointNet++	PointGrid
TYPE	voxelization	native	hybrid
instance average IoU	87.07	85.1	86.4
class average IoU	81.68	81.9	82.2
airplane	88.52	82.4	85.7
bag	86.54	79.0	82.5
cap	74.09	87.7	81.8
car	80.46	77.3	77.9
chair	91.44	90.8	92.1
earphone	51.81	71.8	82.4
guitar	92.61	91.0	92.7
knife	89.44	85.9	85.8
lamp	82.07	83.7	84.2
laptop	96.07	95.3	95.3
motor	65.36	71.6	65.2
mug	92.99	94.1	93.4
pistol	86.72	81.3	81.7
rocket	58.57	58.7	56.9
skateboard	75.70	76.4	73.5
table	85.66	82.6	84.6

et al., 2016): a synthetic dataset with 16,000 point clouds of objects from 16 categories, each of which contains 2 to 6 parts. The objective of the task is to segment the point clouds into one of 50 possible part annotations.

Results. Our results (Tab. 2) demonstrate that gridified networks are also able to achieve competitive performance in segmentation tasks, while being significantly more efficient in terms of parameters, compute and memory. This result validates the ability of gridification to handle dense prediction tasks via gridification and de-gridification.

4.4. Efficiency analysis of gridification

Finally, we investigate the scalability properties of gridification. Specifically, we analyze the time and memory consumption of gridified networks during inference on ModelNet40 for point clouds with increasing size, and compare the computation and memory complexity of convolutional operations on grid and point cloud data.

Scaling gridified networks to large point clouds. Fig 8 shows the average time and memory consumption during inference on ModelNet40 for gridified networks and PointNet++. We observe that gridified networks exhibit a much more favorable scalability both in terms of inference time and GPU allocation –linear vs. quadratic– as the input size and number of channels increase. This demonstrates that gridified networks scale much better than native point cloud methods both for larger point clouds and larger networks.

Scaling the receptive field of neural operations. Furthermore, we analyze the scalability properties of gridified networks relative to the size of its receptive fields. As illustrated in Fig. 1), for native point cloud methods the convolutional kernel must be recomputed for all query points in the point cloud. As a consequence, the construction of the convolutional kernels of size K for all query points in a point cloud with N points incurs in $\mathcal{O}(\mathrm{KD})$ memory and time complexity. In contrast, on grid data, we can compute the kernel once and reuse it at all positions. As a result, on a grid, this operation incurs in $\mathcal{O}(\mathrm{D})$ time and memory complexity.

Fig. 9 show the methods' potential to scale up the receptive field of the gridification module without introducing significant computational overhead.

5. Limitations and future work

The resolution of gridification depends on the size of the point cloud. The main limitation of gridification is that the resolution on the grid is directly proportional to the size of point cloud in order to preserve information. This in turn means that the whole gridified architecture must be changed for point clouds of different sizes, even if they represent the same underlying signal. This is in contrast to native point cloud methods, which, due to their continuous nature, are, in principle, able to generalize to point clouds of different sizes as long as these exhibit the same structures.

Towards no information loss. While gridification aims to produce compact grid representations with minimal information loss, our experiments reveal that some information still gets lost in the process. Loosely speaking, it should be possible to create grid representations that do not lose any information by ensuring that the grid representation has at least as many points and as many channels as the source point cloud representation. Gaining richer theoretical understanding of gridification, could therefore lead to grid representations with no information loss either by imposing other requirements on gridification, or by considering different functional families in the gridification process.

Large scale point clouds and global context. While we verify the scalability and efficiency of gridified networks for increasing point cloud sizes, we only carry on experiments on relatively small datasets. In future work, we aim to deploy gridification to large scale datasets. Furthermore, recent works have shown that using global receptive fields in convolutional operations consistently leads to better results across several tasks, even outperforming well-established Transformer architectures (Gu et al., 2021; Knigge et al., 2023; Poli et al., 2023). Due to the computational complexity of native point cloud methods, networks with global context have not been explored for point cloud processing. With gridification this ability becomes computationally feasible. Exploring the effect of global context for point cloud processing is an exciting research direction.

Generative tasks. Gridification opens up the possibility of performing scalable generative tasks on large point clouds. Gridification can directly be extended to generative tasks if we assume that the point-cloud structure is preserved, i.e., if the coordinates of the output and input point clouds are equal. If this is not the case, e.g., for the generation of molecules (Xu et al., 2019; Hoogeboom et al., 2022),

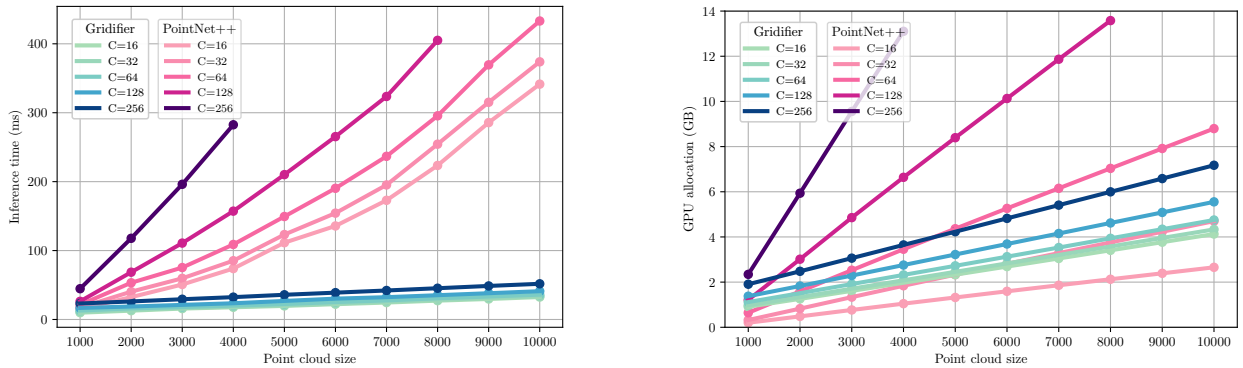

Figure 8. Average time (left) and GPU allocation (right) during inference on ModelNet40 for a batch size of 32.

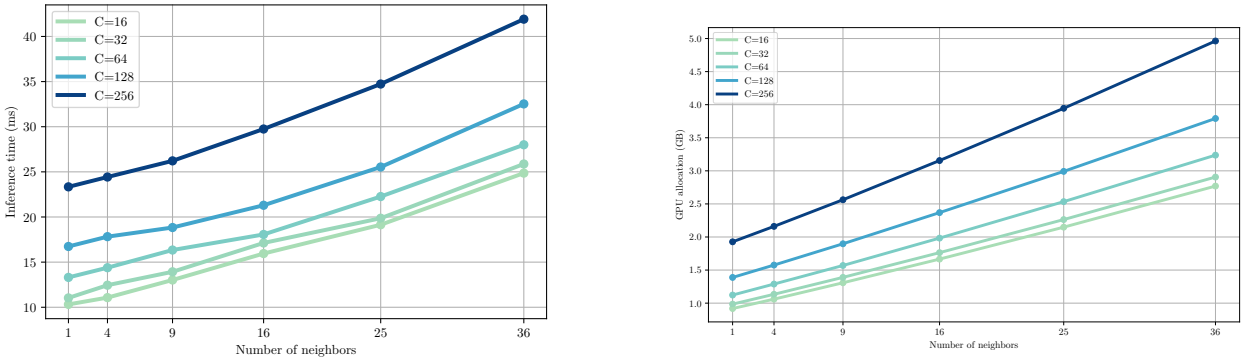

Figure 9. Average time (left) and GPU allocation (right) on ModelNet40 validation set per batch $B = 32$ for various number of neighbors and number of channels C on the grid representation.

de-gridification module must be modified to predict both the features and positions of the new point cloud. We consider this a particularly promising direction for future research.

Equivariant gridification. In its current form, gridified networks do not respect symmetries which might be important for some applications, e.g., equivariance to 3D rotations for the prediction and generation of molecules (Schütt et al., 2017; Hoogeboom et al., 2022). In future work, we aim to extend gridification to respect these symmetries by taking inspiration from equivariant graph neural networks (Fuchs et al., 2020; Satorras et al., 2021). It is important to note that not only gridification and de-gridification must be equivariant, but also that the grid operations in between should respect these properties. This can be achieved in an efficient yet expressive manner through the use of continuous Monte-Carlo convolutions on the regular representations of the group (Finzi et al., 2020; Romero & Lohit, 2022).

6. Conclusion

This work presents gridification, a method that strongly reduces the computational requirements of point cloud processing pipelines by mapping input point clouds to a grid representation, and performing neural operations in there. We demonstrate that gridified networks are able to match the accuracy of native point cloud methods, while being much faster and memory efficient. Through empirical and theoretical analyses, we also show that gridified networks scale much more favorably than native point cloud methods to larger point clouds and larger neighborhoods.

References

Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

Connor, M. and Kumar, P. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE transactions on visualization and computer graphics*, 16(4):599–608, 2010.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Finzi, M., Stanton, S., Izmailov, P., and Wilson, A. G. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *International Conference on Machine Learning*, pp. 3165–3176. PMLR, 2020.

Fuchs, F., Worrall, D., Fischer, V., and Welling, M. Se (3)-transformers: 3d roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems*, 33:1970–1981, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hoogeboom, E., Satorras, V. G., Vignac, C., and Welling, M. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pp. 8867–8887. PMLR, 2022.

Karmakar, N., Biswas, A., Bhowmick, P., and Bhattacharya, B. B. Construction of 3d orthogonal cover of a digital object. In *Combinatorial Image Analysis: 14th International Workshop, IWCIA 2011, Madrid, Spain, May 23-25, 2011. Proceedings 14*, pp. 70–83. Springer, 2011.

Knigge, D. M., Romero, D. W., Gu, A., Gavves, E., Bekkers, E. J., Tomczak, J. M., Hoogendoorn, M., and jakob Sonke, J. Modelling long range dependencies in nd: From task-specific to a general purpose CNN. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ZW5aK4yCRqU.

Le, T. and Duan, Y. Pointgrid: A deep network for 3d shape understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9204–9214, 2018.

Liu, Z., Tang, H., Lin, Y., and Han, S. Point-voxel cnn for efficient 3d deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Maturana, D. and Scherer, S. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 922–928. IEEE, 2015.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.

Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., and Guibas, L. J. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5648–5656, 2016.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017a.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017b.

Ramakrishnan, R., Dral, P. O., Rupp, M., and Von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

Romero, D. W. and Lohit, S. Learning partial equivariances from data. *Advances in Neural Information Processing Systems*, 35:36466–36478, 2022.

Romero, D. W., Kuzina, A., Bekkers, E. J., Tomczak, J. M., and Hoogendoorn, M. Ckconv: Continuous

kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241. Springer, 2015.

Satorras, V. G., Hoogeboom, E., and Welling, M. E (n) equivariant graph neural networks. In *International conference on machine learning*, pp. 9323–9332. PMLR, 2021.

Schütt, K., Kindermans, P.-J., Sauceda Felix, H. E., Chmiela, S., Tkatchenko, A., and Müller, K.-R. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 30, 2017.

Shi, W. and Rajkumar, R. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1711–1719, 2020.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33: 7537–7547, 2020.

Turk, G. and Levoy, M. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 311–318, 1994.

Wang, Z. and Lu, F. Voxsegnet: Volumetric cnns for semantic part segmentation of 3d shapes. *IEEE transactions on visualization and computer graphics*, 26(9):2919–2930, 2019.

Wu, W., Qi, Z., and Fuxin, L. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 9621–9630, 2019.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

Xu, Y., Lin, K., Wang, S., Wang, L., Cai, C., Song, C., Lai, L., and Pei, J. Deep learning for molecular generation. *Future medicinal chemistry*, 11(6):567–597, 2019.

Yi, L., Kim, V. G., Ceylan, D., Shen, I.-C., Yan, M., Su, H., Lu, C., Huang, Q., Sheffer, A., and Guibas, L. A scalable active framework for region annotation in 3d shape collections. *ACM Trans. Graph.*, 35(6), dec 2016. ISSN 0730-0301. doi: 10.1145/2980179.2980238. URL https://doi.org/10.1145/2980179.2980238.

Zhang, C., Wan, H., Shen, X., and Wu, Z. Pvt: Point-voxel transformer for point cloud learning. *International Journal of Intelligent Systems*, 37(12):11985–12008, 2022.

Zhao, H., Jiang, L., Jia, J., Torr, P. H., and Koltun, V. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 16259–16268, 2021.

Appendix

A. Network structure and convolution blocks

Fig. 10 shows how we instantiated the grid network as described in section 2.6 in practice. The `Conv3D` blocks are CCNN blocks as in Knigge et al. (2023).

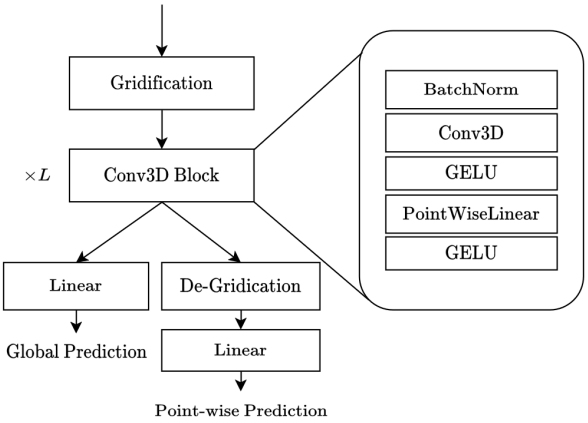

Figure 10. Practical pipeline and convolution blocks.

B. Hyperparameters

Table 3 contains the best hyperparameters for the specific datasets found through hyperparameter sweeps.

Table 3. Hyperparameter settings for the different datasets

	ModelNet40	ShapeNet
batch size	32	16
nr. conv blocks	3	6
hidden channels	128	256
nr. epochs	60	50
nr. input points	1000	2047
Ω position embedding	0.1	1.0
optimizer	AdamW	AdamW
learning rate	0.005	0.001
learning rate scheduler	Cosine Annealing	Cosine Annealing
learning rate warmup	10	10
nr. neighbors	9	9
grid resolution	9	13
conv. kernel size	9	9
dropout	0.1	0.3
weight decay	0	0.001
aggregation	mean	max