LassoNet: Deep Lasso-Selection of 3D Point Clouds

Zhutian Chen, Student Member, IEEE, Wei Zeng, Member, IEEE, Zhiguang Yang, Lingyun Yu, Chi-Wing Fu, Member, IEEE, and Huamin Qu, Member, IEEE

Abstract—Selection is a fundamental task in exploratory analysis and visualization of 3D point clouds. Prior researches on selection methods were developed mainly based on heuristics such as local point density, thus limiting their applicability in general data. Specific challenges root in the great variabilities implied by point clouds (e.g., dense vs. sparse), viewpoint (e.g., occluded vs. non-occluded), and lasso (e.g., small vs. large). In this work, we introduce LassoNet, a new deep neural network for lasso selection of 3D point clouds, attempting to learn a latent mapping from viewpoint and lasso to point cloud regions. To achieve this, we couple user-target points with viewpoint and lasso via a 3D coordinate transform and naive selection, and improve the method scalability via an intention filtering and farthest point sampling. A hierarchical network is trained using a dataset with over 30K lasso-selection records on two different point cloud data. We conduct a formal user study to compare LassoNet with two state-of-the-art lasso-selection methods. The evaluations confirm that our approach improves the selection effectiveness and efficiency across different combinations of 3D point clouds, viewpoints, and lasso selections. Project Website: https://lassonet.github.io

Index Terms—Point Clouds, Lasso Selection, Deep Learning

1 INTRODUCTION

Vast amounts of 3D point clouds have been collected from various sources, such as LiDAR scanning and particle simulation. Exploratory analysis and visualization of point clouds show benefits in many applications, including astronomy, autonomous navigation, and scene reconstruction. Selection is a fundamental task in exploratory analysis of point clouds. However, designing effective selection for 3D point clouds is challenging, especially when the visualization is projected onto a planar 2D surface [19]. The challenge comes from several perspectives: 1) data: a point cloud often consists of a set of unlabeled points, i.e., no information of what label each point holds; 2) visual-
local point density [30, 42], or context-aware that further take into account the lasso location and shape [43]. All methods employ a heuristic that users intend to select regions of higher local point density, which is valid for astronomical datasets where users are typically interested in galaxy cores [42, 43] or halos [30]. However, many other point clouds do not exhibit this property. In this case, these density-based selection techniques lose their advantages and often select unintended clusters that have higher densities of points than target clusters.

In this work, we approach lasso-selection of 3D point clouds from a new angle: lasso-selection is regarded as a latent mapping function \( f \) between point clouds \( (P) \), viewpoint \( (V) \), and lasso \( (L) \), i.e., \( f(P, V, L) \rightarrow P \). We hereby introduce LassoNet, a learning-based approach to seek an optimal mapping based on deep learning techniques. LassoNet integrates deliberated modules to tackle challenges of: (i) data heterogeneity induced by 3D point clouds, viewpoint, and 2D lasso — we associate them using 3D coordinate transformation and naive selection (Sec. 4.1); (ii) generalizability caused by widely ranging number of points and varying point densities — we employ an intention filtering and farthest point sampling (FPS) algorithm (Sec. 4.2). We build a hierarchical neural network to learn local and global features (Sec. 4.3) from a dataset consisting of over 30K lasso-selection records on two different point cloud corpora. Model experiments show that LassoNet can effectively learn the mapping (Sec. 5). We also conduct a formal user study showing that LassoNet advances state-of-the-art lasso-selection methods (Sec. 6).

Our contributions are as follows

- We develop LassoNet — a deep neural network that models lasso-selection of 3D point clouds as a latent mapping from viewpoint and lasso to point cloud regions. To our knowledge, this is a first attempt of learning-based approach that successfully addresses limitations of existing heuristic-based methods.

- We address the challenges of data heterogeneity using 3D coordinate transformation and naive selection, and generalizability using intention filtering and farthest point sampling. We further build a hierarchical neural network to improve network performance.

- We train LassoNet on a new dataset with over 30K lasso-selection records, and release the code and dataset to enable future studies on lasso-selection of point clouds. A formal user study confirms LassoNet fulfills the efficient, effective, and robust requirements.

2 Related Work

Lasso-Selection for 3D Point Clouds. Selection is a fundamental task in interactive visualization [38]. Designing effective interaction theories and methods is regarded as a main challenge for scientific visualization [19]. Systematic reviews of 3D objects selection techniques can be found in [1, 20]. Specifically, lasso-selection is preferable for interacting with 3D point clouds projected on a 2D surface [43]. An ultimate problem here is how to deduce user-intended region in 3D view frustum from a lasso on 2D surface. Cone/Cylinder-lasso [8, 31] is a basic method, which selects all objects within a geometry (i.e., cone or cylinder) extruded from a lasso. The selections can be refined by moving the cone/cylinder [32]. Owhada et al. [26] improved volume data selection by segmenting the data according to user-drawn stroke. This idea of deducing regions of user intention based on underlying data characteristics inspired structure-aware lasso-selection methods [30, 42]. A series of dedicated methods were developed, including TeddySelection and CloudLasso [42]. WYSIWYP (‘what you see is what you pick’) technique [37] can be integrated to provide instant feedback [30]. Recently, context-aware methods [43] that further take into account the lasso position and shape were developed.

Limitations of Existing Lasso-Selection Techniques. Conventional CylinderSelection methods select all the points enclosed by the frustum extruded from an input lasso. In case target points are not located in the same area, Boolean operations of multiple lasso-selections are needed. For instance, it is a common task to select both the left and right wings of an airplane. As illustrated in Fig. 2, users can complete the task by either joining the right and left wing regions with union (top), or removing the body part region from a larger selection with subtraction (bottom). In a more complicated scenario, e.g., to select yellow target points in a more complex scene as in Fig. 1, users need to draw multiple lassos from different viewpoints. Much time will be spent on finding a suitable viewpoint for each target.

CAST [43] have been developed for making 3D point cloud selections more intuitive and efficient. The main idea of CAST family techniques is to infer a user-intended selection region based on properties of point clouds and lasso drawings. They employ a series of heuristics based on density information. Therefore, CAST is particularly effective for selections in 3D point cloud datasets with varying point densities, for instance, astronomy simulations of galaxy collisions and N-body problems. However, not all 3D point clouds, e.g., ShapeNet and S3DIS datasets studied in this work (see Sec. 5), exhibit this property. Taking the airplane (Fig. 2) extracted from ShapeNet for an example, all parts share almost the same point density. CAST hereof will perform similarly to CylinderSelectionL: all points within the frustum extruded from a lasso will be selected.

To make lasso-selections more robust and efficient, we should go beyond scalar properties of point density. One possible solution is to add more intrinsic features of point clouds, such as heat kernel signature [4, 34]. Nevertheless, the approach would need tremendous trial-and-error processes to find suitable parameters (which may not even exist). Instead, we opt to learning-based approach, as we envision that a deep neural network can effectively capture intrinsic features of point clouds, and eventually learn an optimal mapping \( f(P, V, L) \).}

![Fig. 2. To select both two wings of the airplane, two sequential lassos from different viewpoints are required, and selection results are combined using either union (top) or subtraction (bottom) Boolean operations.](Image 293x643 to 345x684)
Deep Learning for Point Clouds. Point cloud is a special type of 3D geometric data that can be processed by deep learning [36]. Based on how to model 3D shapes into CNN processable units, prior researches can be categorized as: Volumetric CNNs (e.g., [25, 39]) convert a 3D shape into voxels and apply a 3D CNN over voxels, which faces a critical problem of sparsity, especially for processing non-uniformly distributed point clouds. Multiview CNNs (e.g., [29, 33]) render 3D shapes into 2D images from multiple viewpoints, and then apply 2D CNNs to analyze them. However, in this work viewpoint is a parameter in the latent mapping that we expect the network to learn. Spectral CNNs (e.g., [3, 24]) learn geometric features defined on 3D manifold meshes, which however, are not easy to construct from 3D point clouds. Many studies (e.g., [7, 10]) have employed feature-based DNNs that convert 3D data into a vector and apply fully connected network to classify the shape. This approach can be seamlessly integrated with shape features, making it popular for processing 3D shapes now.

However, it is extremely challenging to build a suitable DNN for point clouds, as too many features can be derived from enormous points. Qi et al. [27] successfully addressed the challenge with PointNet, which consists of multiple layers of feature transformation, multi-layer perceptron, and max pooling. Based on PointNet, they further developed a hierarchical neural network [28] that improves the prediction accuracy. Recently, a series of follow-up works (e.g., [16, 21, 40]) were conducted including camera position and direction, field of view (FOV), and perspective/orthogonal projection. When a visualization starts, FOV and projection type are usually fixed. Users can control the viewpoint by translating and rotating the camera. Before drawing a lasso on the screen, users tend to find an optimal viewpoint that minimizes occlusion for the target points.

Lasso (L): A lasso L can be represented as an ordered list of points \((y^{\text{scn}}_1 := (u^{\text{scn}}_1, v^{\text{scn}}_1))^{|L|}_{i=1}\), where \(u^{\text{scn}}_i, v^{\text{scn}}_i\) indicate position of a point in a 2D screen space \(\mathbb{R}^2\). The lasso \(L\) meets two requirements: 1) Closed: In case the user-drawn stroke is not closed, we enclose it by connecting its start \((L_1, 0)\) and end \((L_{|L|}, w)\). 2) Non-self-intersecting: In case the input stroke self-intersects, we pick its largest closed part starting and ending at the intersection.

In this work, we regard lasso-selection as a mapping \(f(P, V, L)\) that retrieves a subset of points \(P_s \subseteq P\) based on the current viewpoint \(V\) and lasso drawing \(L\). To be effective, the mapping function \(f\) should minimize difference between \(P_s\) and target points \(P_t\), i.e.,

\[
\begin{align*}
\arg\min_f \{d_s(P_s, P_t) \mid f(P, V, L) \Rightarrow P_s\}
\end{align*}
\]

Meanwhile, we would also like the selection to be efficient, which requires the method should be fast enough for fluid interaction, and robust, i.e., the performance remains effective and efficient over various conditions of point clouds \((P), viewpoints (V),\) and lassos \((L)\).

4 LASSONET
LassoNet pipeline (Fig. 3) consists of three main stages:

1. Interaction Encoding. A primary challenge for this work is implied by data heterogeneity, i.e., to associate viewpoint and lasso information with point cloud. We address the challenge by 3D coordinate transformation that transforms point cloud from object space to camera space, and naive selection that filters a subset of point cloud through CylinderSelection.

2. Filtering and Sampling. The next challenge is to address scalability issue implied by great variability of point clouds (e.g., dense vs sparse) and lasso selection (e.g., small vs large). We employ first an intention filtering mechanism that filters points within an intention area, and then a farthest point sampling (FPS) algorithm that divides dense point clouds into partitions.

3. Network Building. Lastly, we build a hierarchical neural network to learn a latent mapping between point cloud, viewpoint, and lasso. This network structure is inspired by PointNet++ [27, 28] that achieves good performance for segmenting non-uniformly distributed and varying density point clouds.

Fig. 3. LassoNet consists of three stages: In Interaction Encoding stage, we associate point cloud with viewpoint and lasso through 3D coordinate transformation and naive selection; In Filtering and Sampling stage, we reduce the amount of points for network processing through intention filtering and farthest point sampling. Lastly, we build a hierarchical neural network in Network Building stage.

3 Problem Formulation
The scope of this work is constrained to lasso-selection of 3D point clouds using a 2D input device (e.g., mouse) to draw lassos on a 2D surface (e.g., a desktop monitor). Other input and display devices, such as 3D hand gestures and virtual reality HMDs, are out of the scope.

The selection process involves three components:

- **Point cloud** \((P)\): A point cloud \(P\) consists of a set of points \(\{p_{obj}^i := (x_{obj}^i, y_{obj}^i, z_{obj}^i)\}_{i=1}^n\), where \(x_{obj}^i, y_{obj}^i, z_{obj}^i\) indicate position of the point \(p_{obj}^i\) in a 3D object space \(\mathbb{R}^3\). \(n\) is the total number of points, which can be in a wide range from a few thousand (ShapeNet dataset) to hundreds of thousands (3DIS and astronomical datasets). Unlike images that are made up of organized pixel arrays, a point cloud holds no specific order of points. Besides, many point clouds in real-world are unsegmented. We would like the selection method being applicable to them. Nevertheless, the unordered and unsegmented properties compound the difficulty of effective lasso-selection.

- **Viewpoint** \((V)\): A viewpoint \(V\) is determined by many factors, including camera position and direction, field of view (FOV), and perspective/orthogonal projection. When a visualization starts, FOV and projection type are usually fixed. Users can control the viewpoint by translating and rotating the camera. Before drawing a lasso on the screen, users tend to find an optimal viewpoint that minimizes occlusion for the target points.

- **Lasso** \((L)\): A lasso \(L\) can be represented as an ordered list of points \(\{y^{\text{scn}}_1 := (u^{\text{scn}}_1, v^{\text{scn}}_1))_{i=1}^{|L|}\), where \(u^{\text{scn}}_i, v^{\text{scn}}_i\) indicate position of a point in a 2D screen space \(\mathbb{R}^2\). The lasso \(L\) meets two requirements: 1) Closed: In case the user-drawn stroke is not closed, we enclose it by connecting its start \((L_1, 0)\) and end \((L_{|L|}, w)\). 2) Non-self-intersecting: In case the input stroke self-intersects, we pick its largest closed part starting and ending at the intersection.

In this work, we regard lasso-selection as a mapping \(f(P, V, L)\) that retrieves a subset of points \(P_s \subseteq P\) based on the current viewpoint \(V\) and lasso drawing \(L\). To be effective, the mapping function \(f\) should minimize difference between \(P_s\) and target points \(P_t\), i.e.,

\[
\begin{align*}
\arg\min_f \{d_s(P_s, P_t) \mid f(P, V, L) \Rightarrow P_s\}
\end{align*}
\]

Meanwhile, we would also like the selection to be efficient, which requires the method should be fast enough for fluid interaction, and robust, i.e., the performance remains effective and efficient over various conditions of point clouds \((P), viewpoints (V),\) and lassos \((L)\).

3 Problem Formulation
The scope of this work is constrained to lasso-selection of 3D point clouds using a 2D input device (e.g., mouse) to draw lassos on a 2D surface (e.g., a desktop monitor). Other input and display devices, such as 3D hand gestures and virtual reality HMDs, are out of the scope.

The selection process involves three components:

- **Point cloud** \((P)\): A point cloud \(P\) consists of a set of points \(\{p_{obj}^i := (x_{obj}^i, y_{obj}^i, z_{obj}^i)\}_{i=1}^n\), where \(x_{obj}^i, y_{obj}^i, z_{obj}^i\) indicate position of the point \(p_{obj}^i\) in a 3D object space \(\mathbb{R}^3\). \(n\) is the total number of points, which can be in a wide range from a few thousand (ShapeNet dataset) to hundreds of thousands (3DIS and astronomical datasets). Unlike images that are made up of organized pixel arrays, a point cloud holds no specific order of points. Besides, many point clouds in real-world are unsegmented. We would like the selection method being applicable to them. Nevertheless, the unordered and unsegmented properties compound the difficulty of effective lasso-selection.

- **Viewpoint** \((V)\): A viewpoint \(V\) is determined by many factors, including camera position and direction, field of view (FOV), and perspective/orthogonal projection. When a visualization starts, FOV and projection type are usually fixed. Users can control the viewpoint by translating and rotating the camera. Before drawing a lasso on the screen, users tend to find an optimal viewpoint that minimizes occlusion for the target points.

- **Lasso** \((L)\): A lasso \(L\) can be represented as an ordered list of points \(\{y^{\text{scn}}_1 := (u^{\text{scn}}_1, v^{\text{scn}}_1))_{i=1}^{|L|}\), where \(u^{\text{scn}}_i, v^{\text{scn}}_i\) indicate position of a point in a 2D screen space \(\mathbb{R}^2\). The lasso \(L\) meets two requirements: 1) Closed: In case the user-drawn stroke is not closed, we enclose it by connecting its start \((L_1, 0)\) and end \((L_{|L|}, w)\). 2) Non-self-intersecting: In case the input stroke self-intersects, we pick its largest closed part starting and ending at the intersection.

In this work, we regard lasso-selection as a mapping \(f(P, V, L)\) that retrieves a subset of points \(P_s \subseteq P\) based on the current viewpoint \(V\) and lasso drawing \(L\). To be effective, the mapping function \(f\) should minimize difference between \(P_s\) and target points \(P_t\), i.e.,

\[
\begin{align*}
\arg\min_f \{d_s(P_s, P_t) \mid f(P, V, L) \Rightarrow P_s\}
\end{align*}
\]

Meanwhile, we would also like the selection to be efficient, which requires the method should be fast enough for fluid interaction, and robust, i.e., the performance remains effective and efficient over various conditions of point clouds \((P), viewpoints (V),\) and lassos \((L)\).
4.1 Interaction Encoding

To couple point clouds with viewpoint and lasso information is a prerequisite before we can train a DNN model. We achieve this in two steps, as illustrated in Fig. 3(b):

1. **3D Coordinate Transformation**: The viewpoint is determined by many factors, including camera position and direction, FOV, projection type, etc. The information not available in point clouds, from which we only know point positions in 3D object space $\mathbb{R}_{\text{obj}}^3$.

   We encode viewpoint information by transforming coordinates of all point clouds from $\mathbb{R}_{\text{obj}}^3$ to camera space $\mathbb{R}_{\text{cam}}^3$. The transformation can be computed following the graphics pipeline. Specifically, when a user draws the lasso, we record the current position and rotation of the camera, forming a $4 \times 4$ projection matrix. We then derive position of a point in the camera space $p_{\text{cam}}'$ by multiplying the projection matrix with original position $p_{\text{obj}}'$.

2. **Naive Selection**: The next question is how to associate a lasso with the point cloud. Notice that the lasso $L$ consists of an ordered list of points in 2D screen space $\mathbb{R}_{\text{scn}}^2$, while the point cloud has been transformed to 3D camera space $\mathbb{R}_{\text{cam}}^3$.

   As indicated in Fig. 3(b), we associate lasso information with the point cloud using a naive lasso selection method. First, we extrude a lasso $L$ in the 2D screen space to a frustum $F$ in 3D camera space, based on the current viewpoint and screen parameters. Now both point cloud and lasso are transformed to camera space $\mathbb{R}_{\text{cam}}^3$. Thus we can check if a point $p_{\text{cam}}'$ is located inside $F$. We append a binary value $w'$ to indicate if $p_{\text{cam}}'$ falls in $F$: 1 for inside (red points inside Fig. 3(b)), and 0 for outside (blue points in Fig. 3(b)).

   After these, each point $p'_{\text{obj}}$ is modeled as $p_{\text{cam}} := (x_{\text{cam}}', y_{\text{cam}}', z_{\text{cam}}', w')$, where $x_{\text{cam}}', y_{\text{cam}}', z_{\text{cam}}'$ indicate position in camera space $\mathbb{R}_{\text{cam}}^3$, and $w'$ indicates if the point falls inside the frustum $F$ extruded from a lasso.

4.2 Filtering and Sampling

To cope with varying scales implied by point cloud and lasso selection, one simple approach is to add more neurons of a neural network, i.e., scaling up the network width. This, however, will greatly increase the computation time and wreck interactive response. Instead, we employ a filtering-and-sampling approach as in Fig. 3(c):

1. **Intention Filtering**: We would like to filter out points which users definitely have no intention to select. We employ a heuristic that points distant from the lasso are less intended and can be filtered out. Ideally, the intention can be measured as a parabolic function based on distance to the lasso.

   However, it can be inefficient to compute all point distances to a lasso. Instead, we implement a method that is much simpler, yet gives results as good as the parabolic one. When users draw a lasso on the 2D surface, we first find the lasso’s bounding box (yellow dashed rectangle in Fig. 3(c)). We then expand the box 1.2 times outwards (solid purple rectangle in Fig. 3(c)), since the points slightly outside the lasso could also be the intended targets. Points falling outside the expanded box are filtered out.

2. **Farthest Point Sampling (FPS)**: Intention filtering can reduce the amount of points in a great extent. However, in cases that the point cloud is dense or the lasso selection is big, there can still be too many filtered points to fit into a single GPU memory. To cope with these situations, we further employ a downsampling algorithm of FPS which can reduce the number of points meanwhile effectively preserve the convex hull of filtered points [5]. Here, we iteratively split the filtered points into multiple partitions, with each partition consisting of up to three (FPS) points. All partitions are fed to the network for selection prediction. The predictions of all partitions will be combined together as the final output.

   By these, a point cloud is divided into multiple partitions, each of which consists of a set of sample points $\{p_{\text{cam}}^{i} := (x_{\text{cam}}^{i}, y_{\text{cam}}^{i}, z_{\text{cam}}^{i}, w^{i})\}_{i=1}^{\text{FPS}}$.

4.3 Network Building

Filtering and sampling step ensures the amount of points is manageable by a neural network in real-time. This, however, can greatly affect the prediction accuracy, which is then addressed by a hierarchical design of neural network as shown in Fig. 4. A core component here is PointNet (PN) [27], which directly consumes an unordered list of points and outputs a representative feature vector. As illustrated in Fig. 4(a), PN first employs a group of fully connected (FC) layers to map each point into a high-dimensional space. The FC layers share parameter weights to reduce the number of parameters and accelerate network convergence. Finally, a pooling layer is used to aggregate the high-dimensional features and output a feature vector that can be regarded as the signature of the input point cloud.

However, the pooling layer in PN only remains the global information of the whole point cloud. The relationship between a point and its local neighborhood is missing, which is not desired since it decreases the prediction accuracy. Alternatively, we employ abstraction and propagation hierarchical structures as in PointNet++ [28] to generate features of both local and global information.
We choose two point cloud corpora that have been widely used in many applications, namely, ShapeNet (left) and S3DIS (right): target and interfering points are colored in yellow and blue respectively, while lassos are in red color.

- **Abstraction** (Fig. 4(b)): We first divide the whole input points into several groups of equal size. Each point group is represented as a level-0 feature vector. We then apply PN to each point group, yielding a level-1 feature vector characterizing local features for each group of points. Each feature vector can actually be modeled as a set of high-dimensional points, which can again be processed by PN to extract correlations among point groups. This grouping and correlation extraction processes are recursively repeated \( k \) times. By this, we obtain a level-\( k \) feature vector that stores both global and local features of the input point cloud.

- **Propagation**: The next challenge is how to propagate the level-\( k \) feature vector to individual points. Here, we first concatenate level-\( k \) with level-\((k-1)\) feature vectors, and applying a FC layer that generates a propagated layer of feature vector. The process is again repeated \( k \) times until each point group is propagated. By this, we generated a final feature vector containing rich information for each point, including its local relation to neighboring points, and its global relation to the whole point cloud.

Based on the final feature vector, we can predicate for each sample point \( p_{\text{cam}} \) a probability value \( \rho \) using FC layers with a softmax function, which falls in \([0, 1]\) indicating the probability that \( p_{\text{cam}} \) is selected. If \( \rho \) is larger than 0.5, we regard \( p_{\text{cam}} \) as selected; otherwise not.

5 **MODEL EXPERIMENTS**

To train LassoNet, we collect a dataset of more than 30K lasso-selection records (Sec. 5.2) annotated on two publicly available point cloud corpora (Sec. 5.1). Then we introduce the training process (Sec. 5.3), and finally report the quantitative evaluation results (Sec. 5.4).

5.1 Point Cloud Preparation

We choose two point cloud corpora that have been widely used in many applications, e.g., robotics and scene reconstruction. The first one is ShapeNet [41], containing in total over 16K point clouds of CAD models in 16 categories (e.g., airplane, car, bag). Each point cloud consists of several thousand points and point densities from 10K to 6M points/m\(^2\). The points are divided into two to six parts, e.g., an airplane is divided into body, wing, engine, and tail. The second corpus is Stanford Large-Scale 3D Indoor Spaces (S3DIS) dataset, containing 272 point clouds collected by high-resolution 3D scanning of rooms. The point clouds exhibit a wide range of point densities from 60K to 3M, and point densities ranging from 0.2K to 60K points/m\(^2\). The points are also divided into parts, e.g., chair, table, and floor.

To improve the quality of annotation, we first filter out points in occluding the view heavily, e.g., ceiling points in S3DIS datasets. Points are also divided into parts, e.g., chair, table, and floor. 3M, and point densities ranging from 0.2K to 60K points/m\(^2\).

5.2 Lasso-Selection Annotation

We recruit 20 professional annotators to generate lasso selection records on the point cloud corpora. The annotation is done on a web-based visualization platform that renders target points in yellow and interfering points in blue with a fixed FOV of 60° (see Fig. 5). For each point cloud, we randomly allocate one part (e.g., wings of an airplane, or a table in a room) as the target, and the others as interfering points. The platform supports 5-DOF navigation using mouse input.

The annotators are asked to enclose target points by drawing an appropriate lasso (see red lassos in Fig. 5) from a good viewpoint. Then, the target points inside of the lasso are highlighted to indicate how successful the selection is. Thus, for each set of target points, no matter the points are separated or not, we allow for only one lasso selection. Taking the airplane in Fig. 5 for an example, to select both wings, users are allowed to draw a lasso from different viewpoints as in the first two subfigures, but not to draw two lassos. We encourage the annotators to complete the selection as good as possible, so we do not set an explicit time limit in the annotation. When an annotation is finished, a backend process will record information of point cloud id, target points ids, current camera position & direction, and lasso drawings. To ensure annotation quality, we clean up records that cover less than 70% of the target points or more than 80% non-target points.

5.3 Network Training

Following conventions in machine learning, we randomly split annotations records by point clouds into 9:1 for training and testing. In this way, point clouds for testing do not appear in the training set. This yields 2,092 out of 2,332 point clouds from ShapeNet, and 242 out of 272 from S3DIS for training.

**Loss function.** Since our task can be formulated as a per-point binary classification problem (i.e., selected vs non-selected), we adopt a cross entropy loss function to train LassoNet. For a training record, we calculate the loss on each point and then average the losses over all points to update the network by a backward propagation algorithm. The loss for each training record can be calculated as:

\[
L = -\frac{1}{n} \sum_{i=1}^{n} \left( \theta_0 \rho_i \log(\rho_i) + \theta_1 (1 - \rho_i) \log(1 - \rho_i) \right),
\]

where \( n \) is the number of points in a training point cloud \( P := \{ \rho_i \}_{i=1}^{n} \).

\( \rho_i \) is a binary value indicating the ground-truth status of a point \( \rho_i \): 0 for interfering points, and 1 for target points. \( \rho_i \) is the probability value of \( p_i \) predicted by LassoNet. To improve robustness of LassoNet on point clouds with extremely imbalanced numbers of target and interfering points, we add \( \theta_0 \) and \( \theta_1 \) to control weights of the two classes. Specifically, the interfering points are usually much more than target points in S3DIS annotations, thus we set \( \theta_0 = 4 \) and \( \theta_1 = 1 \). In contrast, \( \theta_0 \) and \( \theta_1 \) are both set to 1 in ShapeNet.

**Hyper parameters.** There are two hyper parameters that play important roles in LassoNet, namely, threshold of FPS (FPS), and size of a group \( \text{size}(g) \) in network building (Sec. 4.3).
where 0 indicates best performance for worst case performance. Both $d_J$ and $F_1$ evaluations from the following perspectives:

### Implementation Details

T $\ell_1$-regularization of the network, ranging from 1 to three FPS. A smaller $size(g)$ makes the network focus more on local features of a point cloud, but leads to deeper hierarchy and more computational cost. A bigger $size(g)$ allows the network to compute more efficiently, but less accurate predictions caused by the lack of sufficient local details. $size(g)$ should be set based on the characteristics of the target datasets. Empirically, we set $size(g)$ to 2048 for ShapeNet annotations, since the point clouds contain only a few thousand points. For S3DIS annotations, we set $size(g)$ to 32 that strikes a good balance between effectiveness and efficiency.

### 5.4 Evaluation

#### Accuracy performance

Accuracy performance is a main criterion for lasso selection techniques. As discussed in Sec. 3, the difference between selection points $P_t$ and target points $P_s$ should be minimized. We measure the difference using Jaccard distance, which is calculated as:

$$d_J(P_t, P_s) = 1 - \frac{|P_t \cap P_s|}{|P_t| + |P_s| - |P_t \cap P_s|}$$

We further include $F_1$ score that is often used in measuring binary classification performance. $F_1$ is measured upon true positive ($TP = |P_t \cap P_s|$), false positive ($FP = |P_t - P_s \cap P_s|$), and false negative ($FN = |P_s - P_t \cap P_s|$): $F_1 = 2TP/(2TP + FP + FN)$. In general, $F_1$ score tends to measure average performance, while $d_J$ tends to measure the worst case performance. Both $d_J$ and $F_1$ are in the range of $[0, 1]$, where 0 indicates best performance for $d_J$ but worst performance for $F_1$, and vice versa.

We compare LassoNet with CylinderSelection - a basic lasso selection method for 3D point clouds. Table 2 presents the comparison results on the testing annotations from ShapeNet and S3DIS separately. Overall, LassoNet achieves much better performance than CylinderSelection on both annotation datasets in terms of both $F_1$ score and $d_J$. Specifically, we notice that the performance of CylinderSelection drops much on S3DIS annotations, while LassoNet only drops a bit. We hypothesize this is because S3DIS annotations are more diverse than ShapeNet annotations. To validate the hypothesis, we conduct further evaluations from the following perspectives:

- **Scene complexity.** We quantify scene complexity using the number of parts in a point cloud. Point clouds in ShapeNet contain a limited number of parts ($\leq 6$), while S3DIS point clouds usually consist of tens of parts. Figure 6 compares CylinderSelection and LassoNet over variations of scene complexity. On the left, average $d_J$ are measured for ShapeNet annotations divided into groups of $[0, 10), [10, 20), [20, 30), [30, 40), and [40, +\infty]$ parts. Again, $d_J$ for CylinderSelection remain high in all cases, while LassoNet remains low around 0.2. Surprisingly, we notice that the number of parts exceeds 40, $d_J$ of CylinderSelection drops, while LassoNet increases.

- **Task complexity.** We quantify task complexity using the percentage of target points in a point cloud. Big targets (i.e., higher percentage) are typically easier to select than small ones (i.e., small percentage). Figure 7 shows the comparison results. Targets in ShapeNet are occupying higher percentages than those in S3DIS. We divide ShapeNet annotations into 5 equal ranges, and measure the average $d_J$ as presented on the right. As expected, $d_J$ of both CylinderSelection and LassoNet drop when the percentage of target points increase. The same trend can be observed for CylinderSelection on S3DIS annotations, as shown Fig. 7(left). Since the scene is much complex, we divide the annotations according to target point percentages in the range of $[0, 1\%], [1\%, 2\%], [2\%, 3\%], [3\%, 4\%], [4\%, +\infty]$. In contrast, we can see that LassoNet achieves stable and better performances across the five groups.

### Time performance

Time performance is another criterion for lasso selection techniques. Table 2 also presents a comparison of time costs for CylinderSelection and LassoNet on ShapeNet and S3DIS annotations. Here, CylinderSelection is implemented in WebGL with average time costs of 16.67ms for ShapeNet and 18.86ms for S3DIS. LassoNet requires additional times for network computation, which adds up to 20.47ms and 69.46ms for ShapeNet and S3DIS, respectively. The increments are reasonable given that point clouds in S3DIS contains hundreds of times more points than those in ShapeNet. The time costs are also comparable with state-of-the-art lasso selection methods such as CloudLasso [42]. Nevertheless, time costs for accomplishing accurate selection tasks of LassoNet are actually less than those of CylinderSelection and CAST; see Fig. 10 and Sec. 6.3 for details.
Underfitting and overfitting are critical challenges in machine learning [9]. Underfitting occurs when the model cannot fit the training set, while overfitting occurs when the model fits the training set well but fails to fit the testing set. In network training stage, we have adopted multiple strategies, including dropout, weight decay, and batch normalization, to avoid the issues. Nevertheless, to further investigate whether these issues occur, we examine $d_j$ per epoch in the training and testing processes, which are plotted as blue and red lines as shown in Fig. 8, respectively. From the figures, we can notice that $d_j$ in training process decreases rapidly and smoothly, and $d_j$ in the testing process also decreases with a small gap between that in the training process. The observations confirm that our model does not suffer from underfitting and overfitting problems.

6 USER STUDY

In reality, users typically complete selection of target points using a sequence of lassos. To cope with this fact, we conduct a formal user study to further evaluate the performance of LassoNet in comparison with two lasso-selection methods of conventional CylinderSelection and SpaceCast – a state-of-the-art density-based selection technique. SpaceCast is chosen since it is the only method in the CAST family that is able to select a part of the cluster in case there is no density variation (such as two wings of an airplane). Here, we allow users to refine a selection using Boolean operations of union, intersection, and subtraction for all three interactions.

This section reports quantitative results of the study in terms of efficiency measured as completion time, and effectiveness measured as Jaccard distance. By comparing efficiency and effectiveness over different datasets, we further evaluate robustness of LassoNet.

6.1 Experiment Design

Participants: We recruited 16 participants (9 males and 7 females) in the study. 13 participants are students from different disciplines such as computer science and biochemistry, while the other three are research staff. All participants had at least a Bachelor’s degree. The age of the participants ranges from 22 to 29, with the mean age of 24.69 years ($SD = 1.78$). All participants reported to be right-handed. Four participants had experience of working with point clouds. Three participants had experience of manipulating 3D objects, and they are familiar with basic 3D interactions, including rotation and zoom-in/out. All participants completed the experiments in about 90 minutes.

Apparatus and Implementation: Testing datasets from ShapeNet and S3DIS were converted into a data format of point positions. A web-based visualization is developed for LassoNet, while CylinderSelection and CAST are running on CAST application developed by the authors. To eliminate bias caused by rendering effects, we adopted the same settings of FOV, background, and point colors with the CAST tool. Target points were rendered in orange color while interfering points were marked in red. CylinderSelection and SpaceCast, on D3, while better performances on D1 and D2 which do not have varying point density. Moreover, since CylinderSelection does not require any additional computation, we expected that CylinderSelection would achieve similar performance with SpaceCast on D3 (H1.2).

H1: LassoNet would be more efficient, i.e., less completion time, than CylinderSelection on all datasets (H1.1). Compared with SpaceCast, LassoNet would be more efficient on D1 & D2, while equally efficient on D3 (H1.2).

H2: LassoNet would be more effective, i.e., smaller Jaccard distance, than CylinderSelection on all datasets (H2.1). Compared with SpaceCast, LassoNet would be more effective on D1 & D2, while being equally efficient on D3 (H2.2).

H3: CylinderSelection would be the most robust, i.e., similar completion times (H3.1) and Jaccard distances (H3.2), on all three datasets. LassoNet would be more robust than SpaceCast.

Task and Procedure: The task was to select target points marked in orange while avoiding interfering points marked in blue. Selected points would be marked in red. In CylinderSelection and SpaceCast, the participants were allowed to refine the selections by three Boolean operations: union, intersection, and subtraction, in case they were not satisfied by the results. The participants were reminded that completion time is also an evaluation metric. So they were expected to complete the tasks as soon as possible in case they were satisfied with the results. The participants could take a 5-minute break when they felt tired.

Before actual experiments, we explained to the participants about the principles of the next lasso-selection method. We demonstrated how to change the viewpoint, draw lassos, and select the target points on the screen. To ensure the participants fully understood the interactions, they were asked to practice with three training datasets. In the training trials, we gave the participants as much time as they needed. To suppress learning effects gained from previous assignments, we assigned a sequence of lasso-selection methods pseudo-randomly to each participant. When participants felt satisfied with the results, they proceeded to the next task by pressing a Submission button, and a back-end process automatically recorded the completion time and accuracy for the current task. In the end, the participants were asked to complete a questionnaire for user feedback on their satisfactory of each method.

D1 D2 D3

Fig. 9. Exemplar point clouds employed in the user study.

Datasets: We conducted the user study on three different datasets, as shown in Fig. 9. Besides ShapeNet (denoted as D1) and S3DIS (D2) described in Sec. 5.1, we recruited a third dataset of astronomy point clouds that were used in CAST experiments (D3). D3 consists of four point clouds, each of which is made up of 200K to 400K points representing multiple particle clusters. The clusters have equal uniform densities and are surrounded in a low-density noise environment. The target cluster was located either in the center or was partially surrounded by interfering points so that it is tricky to find a clear view to the whole target. We trained a new model for D3, using only 600 lasso-selection records manually annotated by ourselves. The other settings are the same as those used when training S3DIS annotations. Same as [43], participants were asked to select some of the small clusters. From each dataset, we selected three different point clouds with one meaningful part as target points. All the point clouds and target points were not used for training. In total, there were 9 assignments (3 point clouds × 3 targets) for participants to complete using each method.

Fig. 10. Comparison of completion time (left) and Jaccard distance (right) for CylinderSelection, SpaceCast, and LassoNet on three datasets.
6.3 Quantitative Results

We collected in total 432 records (16 participants × 3 techniques × 9 assignments) from the user study. Figure 10 presents a comparison of mean completion time (left), Jaccard distances (right), and their 95% confidence intervals for each technique conducted on each dataset. At noticed, LassoNet outperformed CylinderSelection on both completion time and accuracy. LassoNet also achieves better performance than SpaceCast on D1 & D2, and similar performance on D3.

We performed a two-way ANOVA (3 techniques × 3 datasets) on both completion time and Jaccard distance. Before the analysis, we first confirmed that all results of completion time and Jaccard distance in each condition follow normal distribution using a Shapiro-Wilk test. Prerequisites for computing ANOVA are fulfilled for the hypothesis. All hypotheses are confirmed by the analyses. Below we report details of individual analysis.

**Completion Time.** As expected, *selection technique* had a significant effect on completion times ($F_{2,429} = 82.73, p < 0.0001$). Average completion times (Fig. 10(left)) are 37.34s for CylinderSelection (SD = 18.74), 27.14s for SpaceCast (SD = 17.72), and 15.49s for LassoNet (SD = 7.81). Post-hoc tests using Bonferroni correction indicate that LassoNet is significantly faster than both CylinderSelection ($p < 0.0001$) and SpaceCast ($p < 0.0001$). Through more detailed probes, we found that LassoNet was significantly faster than CylinderSelection ($F_{1,47} = 69.62, p < 0.0001$), ($F_{1,47} = 61.51, p < 0.0001$), ($F_{1,47} = 39.60, p < 0.0001$) on D1, D2, and D3, respectively. This confirms the hypothesis H1.1. We also found that LassoNet was significantly faster than SpaceCast ($F_{1,47} = 72.28, p < 0.0001$) on D1, and ($F_{1,47} = 28.28, p < 0.0001$) on D2. No significant difference is observed for LassoNet and SpaceCast on D3 ($F_{1,47} = 2.30, p = 0.13$). This further confirms the hypothesis H1.2.

We checked effects of *dataset* on completion time. Using Bonferroni correction test, we found that *dataset* shields a significant effect on SpaceCast ($F_{2,141} = 31.72, p < 0.0001$), while no significant effect on CylinderSelection ($F_{2,141} = 1.91, p = 0.15$) and LassoNet ($F_{2,94} = 1.91, p = 0.008$). This result confirms the hypothesis H3.1.

**Jaccard Distance.** We repeated ANOVA tests on Jaccard distance (Fig. 10(right)), by which significant effects imposed by selection techniques were observed ($F_{2,429} = 29.77, p < 0.0001$). LassoNet achieved a much lower mean Jaccard distance of 7.65% (SD = 6.94), in comparison with CylinderSelection (mean = 14.44%, SD = 11.61) and SpaceCast (mean = 17.95%, SD = 14.68). LassoNet is significantly more effective than CylinderSelection on all three datasets ($F = 36.38, p < 0.0001$), which confirms the hypothesis H2.1. Compared to SpaceCast, LassoNet is slightly more but not significant ($F=1.87, p=0.18$) error-prone on D3, whilst it is significantly effective on D1 ($F = 72.28, p < 0.0001$) and D2 ($F = 28.28, p < 0.0001$). These results confirm H2.2.

Though not significant, CylinderSelection achieves the most consistent Jaccard distances on different datasets ($F_{2,141} = 3.52, p < 0.05$), in comparison to SpaceCast ($F_{2,141} = 32.60, p < 0.0001$) and LassoNet ($F_{2,141} = 10.73, p < 0.0001$). Nevertheless, LassoNet is more stable than SpaceCast. Hypothesis H3.2 hereof is confirmed.

6.4 Qualitative User Feedback

We also collected qualitative user feedback from the participants after the user study. 13 out of 16 participants prefer LassoNet, due to its simplicity to learn and to use. One participant stated that “a person knowing how to control mouse should feel no difficulty in lasso-selection”. They also appreciated that the visual interface returned immediate feedback upon lass selections. The Boolean operations of union, intersection, and subtraction posed some difficulty for them in the beginning, but they fully understood the operations through several trial-and-error trainings finally. Below we summarize their feedback for each method.

- **CylinderSelection.** All participants felt that results from CylinderSelection are most predictable. Some participants reported that this was highly appreciated because then they can refine the selections using Boolean operations as expectations. However, we also noticed that the participants showed interests to refine selections at the beginning, but the interests dropped quickly after a few assignments. This reaction was particularly obvious on D2, where the scenes are complex so that participants would need to change viewpoints very often when making refinements. This explains why average completion time of CylinderSelection is slightly less on D2 than on D1.

- **SpaceCast.** Three participants expressed high praise for SpaceCast on D3, which allowed them to make pretty accurate selections. In fact, most participants would choose SpaceCast as their favorite method if the experiments were conducted on D3. However, all participants felt that SpaceCast was very unpredictable on D1 & D2, even though we had clearly explained the underlying mechanism. Often the results were only a part of what they intended to select. For instance, when the assignment was to select the left wing of an airplane (see Fig. 2), SpaceCast often selected only the engine part. We suspect the reason was that the engine has a slightly higher density of points than the wing.

- **LassoNet.** Most participants favored LassoNet since the selections best match with their intention. “It seems the method can really understand what I want”, one participant commented. Nevertheless, the participants also figured out that refinement using LassoNet is not as feasible as CylinderSelection. When making refinements, participants often select only a few points at a very close view. In such scenarios, LassoNet tends to select more points that exhibit strong correlations with the target points (e.g., neighboring, symmetric, etc.), see Fig. 12 for an example. They suggested adding Boolean operations in the technique for refinements.

7 Discussion

7.1 Examples

Figure 1 presents a typical example of selection task: in a complex room, users need to three regions of different objects. For conventional CylinderSelection method, users need to adjust viewpoints according to the current region of selection, and also need to refine selections using Boolean operations. Instead, LassoNet can complete the task directly
We also tried to train a single model for both ShapeNet and S3DIS. As for now, employing different parameter settings for different datasets (probably) identifies similarity features among the chairs and selects all annotations, yielding an average for target data that exhibit different but related distributions with source data. Moreover, domain adaptation can also learn a well-performing model for point cloud processing, such as MCCNN [14], also trained separately is practically more feasible. Many recent deep neural network models are mainly designed for ShapeNet point clouds. Nevertheless, the results still outperform S3DIS point clouds are usually smaller regions, in comparison to those clouds are samples from synthetic CAD models, and 3) selections for densities in the two corpora are very different, as S3DIS point clouds are hundred times greater than those in ShapeNet point clouds.

LassoNet (again probably) detects symmetric features of trailing edge in the left and right airfoils, which is however not desired. LassoNet can also effectively select multiple separate parts by using only one lasso, as illustrated in Fig. 12 (left). Here, a user draws a lasso to enclose all three target chairs, and LassoNet correctly segments points belonging to the chairs from surrounding points. Notice that the training dataset does not include such cases (for S3DIS data, only one object is selected as the target for each annotation). LassoNet (probably) identifies similarity features among the chairs and select all of them together. However, we also notice that sometimes LassoNet gives unexpected results, as shown in Fig. 12 (right). We assume that LassoNet (again probably) detects symmetric features of trailing edge in the left and right airfoils, which is however not desired.

7.2 Generalizability

We also tried to train a single model for both ShapeNet and S3DIS annotations, yielding an average $d_1$ of 0.214 and $F_1$ score of 0.873. The performance drops in comparison with those achieved by LassoNet using separate models (see Tab. 2). The cause for performance dropping mainly comes from large differences between ShapeNet and S3DIS annotations: 1) numbers of points in S3DIS point clouds are almost a hundred times greater than those in ShapeNet point clouds, 2) point densities in the two corpora are very different, as S3DIS point clouds are collected by sensing of real-world indoor rooms, while ShapeNet point clouds are samples from synthetic CAD models, and 3) selections for S3DIS point clouds are usually smaller regions, in comparison to those for ShapeNet point clouds. Nevertheless, the results still outperform basic CylinderSelection.

Training a single model for multiple datasets is a challenging task. As for now, employing different parameter settings for different datasets is practically more feasible. Many recent deep neural network models for point cloud processing, such as MCCNN [14], also trained separate models for different datasets. A potential solution is domain adaptation [2], especially multi-source domain adaptation, that has proven beneficial for learning source data from different domains [12]. Furthermore, domain adaptation can also learn a well-performing model for target data that exhibit different but related distributions with source data, thereby improving generalizability of the model. Thus, we consider domain adaptation as an important direction for future work.

7.3 Limitations and Future Work

Though the model experiment and user study have demonstrated that LassoNet advances prior methods, there are several limitations.

- All deep neural network models, no matter supervised or unsupervised, require tremendous amounts of training data to generate high prediction accuracy. We tackle this issue using a new dataset with over 30K records generated by professional annotators. It is also feasible to extend this dataset by synthesizing variations from existing records [6]. Yet, there can still be certain scenarios not covered by the training data.

- When making refinements, users would like to select only a few points. As observed by the participants, LassoNet tends to expand the selection to some closely correlated points. A feasible solution here would be to add a conditional statement in LassoNet: when naive selection detects only a few points being selected, LassoNet automatically returns these points as output.

We also identify several promising direction for future work:

- A first and foremost work is to update our backbone network to state-of-the-art deep learning models for processing point clouds. For example, Hermosilla et al. proposed MCCNN that utilizes Monte Carlo up- and down-sampling to preserve the original sample density, making it more suitable for non-uniformly distributed point clouds [14]. We consider MCCNN as an important future improvement to enhance the effectiveness and robustness of LassoNet.

- Second, we would like to develop visual analytics to get a better understanding of what has the network learned, which currently is a ‘blackbox’. As for now, we suspect that the network has modeled several intrinsic properties of point clouds, including i) local point density, as astronomic point clouds exhibit; ii) symmetric property, as indicated by airplane wings; iii) heat kernel signature [4, 34], as the network can segment seat and legs of a chair (Fig. 11 (left)). The issue calls for more visual analytics to ‘open the black box’ [15, 22].

- Last but not least, we would like to incorporate more parameters in the mapping function, such as FOV and stereoscopic projection. Up to this point, we have only modeled viewpoint in terms of camera position and direction, but not other parameters. Modeling these parameters would be necessary and interesting, as it can potentially extend the applicability of LassoNet to many other scenarios, e.g., to improve selection in VR/AR environments (e.g., [17, 35]).

8 Conclusion

We presented LassoNet, a new learning-based approach of lasso-selection for 3D point clouds built upon deep learning. Our approach can be readily applicable to any scenario where one has a set of unordered points ($P$), a 2D surface for visualizing the points ($V$), and a lasso on the surface ($L$). Essentially, LassoNet can be regarded as an optimization process of finding a functional latent mapping function $f(P,V,L) \rightarrow P_s$ such that $P_s$ matches best with a user’s intention of selection $P_t$. To learn such an optimal mapping, we created a new dataset with over 30K selection records on two distinct point cloud corpora. LassoNet also integrates a series of dedicated modules including coordinate transformation, intention filtering, and furthest point sampling to tackle the challenges of data heterogeneity and scalability. A quantitative comparison with two prior methods demonstrated robustness of LassoNet over various combinations of 3D point clouds, viewpoints, and lassos in terms of effectiveness and efficiency.

Acknowledgments

The authors wish to thank the anonymous reviewers for their valuable comments. This work was supported in part by National Natural Science Foundation of China (No.61802388 and No.61602139). This work is partially supported by a grant from MSRA (code: MRA19EG02).