Learning Point Clouds in EDA

(Invited Paper)

Wei Li
Chinese University of Hong Kong
wli@cse.cuhk.edu.hk

Guojin Chen
Chinese University of Hong Kong
glchen@cse.cuhk.edu.hk

Haoyu Yang
Chinese University of Hong Kong
hyyang@cse.cuhk.edu.hk

Ran Chen
Chinese University of Hong Kong
rchen@cse.cuhk.edu.hk

Bei Yu
Chinese University of Hong Kong
byu@cse.cuhk.edu.hk

Abstract

The exploding of deep learning techniques have motivated the development in various fields, including intelligent EDA algorithms from physical implementation to design for manufacturability. Point cloud, defined as the set of data points in space, is one of the most important data representations in deep learning since it directly preserves the original geometric information without any discretization. However, there are still some challenges that stifle the applications of point clouds in the EDA field. In this paper, we first review previous works about deep learning in EDA and point clouds in other fields. Then, we discuss some challenges of point clouds in EDA raised by some intrinsic characteristics of point clouds. Finally, to stimulate future research, we present several possible applications of point clouds in EDA and demonstrate the feasibility by two case studies.

1 Introduction

The exploding of deep learning techniques have motivated the development of intelligent EDA algorithms from physical implementation to design for manufacturability. Among various deep learning based methods, the study on irregular data is one of the core subjects in enormous EDA problems and optimization algorithms. Irregular data is particularly good at modeling pairwise relationships among different discrete items and preserving original features without any information loss, thereby making it a crucial topic in EDA considering that most underlying structures are also organized in a non-Euclidean space.

In the history of EDA development, a considerable attention of irregular data has been paid to graph, a classical abstract representation that has been widely investigated over the past few decades by extensive elegant graph algorithms and graph-based learning approaches. Recently, benefited from the presence of a large amount of data and the fact that graph is a natural and powerful representation for many fundamental objects in EDA applications, some deep learning-based attempts on graphs are made in the EDA field. These efforts have demonstrated the overwhelming representation power of graph when representing some basic elements in EDA applications such as netlist [1–7] and layout [8]. To solve the irregularity of the graph, most works adopt a convolution-like operation referred to graph convolution. The graph convolution works in a message-passing way between neighbors, and thus makes it flexible with graph structures that are unordered and variable in size.

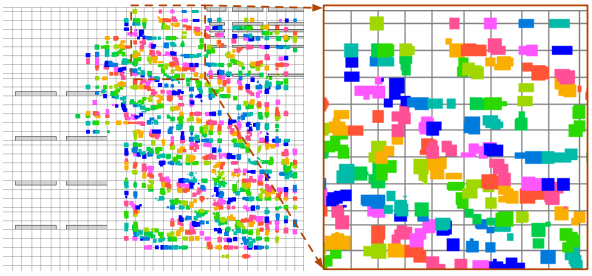


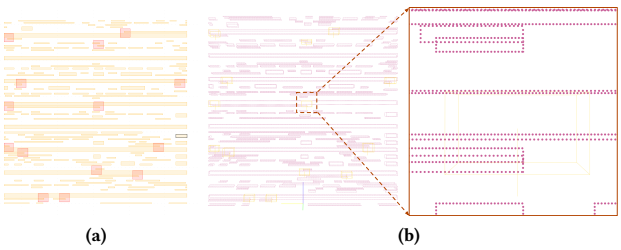
Figure 1: An example of the result after placement, originally shown in the slides of [9]. The standard cells marked by different colors can be regarded as scattering point set.

However, graph is not appropriate for some data formats in EDA since it strictly constrains inter-connected relationships. That is, graph construction requires the definition of connections (edges) among objects (nodes) to leverage the graph abstraction, which brings about unexpected information bias. For example, the pin set for the tree construction does not contain connection information between any two pins, and a graph abstraction of the pin set may even mislead the construction result. In these cases, point cloud, defined as a set of data points in space, is a better representation since it directly preserves the original geometric information without any discretization or misinterpretation.

To provide insights into point clouds, we summarize two viable (but not limited) application directions of point clouds in EDA. First, it is interesting to see that point clouds, i.e., the sets of points, have already existed in many EDA problems as the fundamental structure of input. For example, both routing tree construction and clock tree construction are to build a tree based on a point cloud including the sink pins and the root pin. Another potential application happens after placement: some predictions or estimations after placement can view the placement output (shown in Figure 1) as a point cloud as long as the interconnection between cells plays a trivial role in these tasks. Moreover, point clouds can be used to represent solid objects in a finite element analysis context. In other words, a point cloud representation boils down mathematically complex

ISPD '21, March 22–24, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8300-4/21/03…$15.00
https://doi.org/10.1145/3439706.3446895



(a) (b)

Figure 2: Examples of the transformation from layout to point cloud. Figure 2(a) is the original GDSII layout, the hotspot is marked as red rectangle. Figure 2(b) is the transformed point cloud.

geometry into a relatively finite number of points. The layout pattern is a representative case. Conventional representation for the layout patterns is to simply convert the polygon shapes of a layer into an image [10, 11], which suffers from several drawbacks. First, the obtained image may be too large (typically $> 1000 \times 1000$) to be space and computation friendly, thus restricting the scalability of the entire framework. Second, the geometry and topology features of images are usually exceptionally obscure, thereby resulting in possible redundancy. In contrast, point cloud representation resolves both drawbacks. By sampling key points on the boundaries and corners of the layout patterns, not only the information is reserved without redundancy but also the memory consumption is reduced. An example of the point cloud representation of layout is shown in Figure 2, where each layout feature is decomposed to discrete points. In this paper, to stimulate future research, we present two case studies corresponding to the two applications mentioned above ,i.e., the direct transformation and the indirect decomposition.

We organize this paper as follows. Section 2 reviews previous deep learning techniques in EDA by topics. Section 3 introduces previous deep learning based methods for point clouds and discusses the challenges of point clouds in EDA applications. Section 4 and Section 5 present the case studies of routing tree construction and hotspot detection respectively. Finally, we conclude the paper in Section 6.

2 Deep Learning in EDA

In this section, we briefly cover previous deep learning techniques in EDA by topics, i.e., routing, placement, and lithography. In addition, we discuss the applications of Graph Neural Networks in EDA in an independent subsection because graph is closely related to point clouds as another widely-used irregular data representation.

2.1 Convolutional Neural Networks in EDA

Routability estimation in early stage has been demonstrated necessary to enable fast design closure. RouteNet [12] builds up fully-convolutional neural networks for both design rule violation count prediction and DRC hotspot prediction. An 18-layer ResNet is adopted as the backbone structure that takes the input of a 3D tensor containing the information of pin density, macro region, RUDY and RUDY pins. Recently, Liang *et al.* [13] takes advantage of U-Net architecture to predict routing congestion location with the image-based placement information. There is also investigation of fully convolution networks for routing congestion prediction based on global placement results [14], which is demonstrated efficient on industrial designs.

Besides the routability related tasks, deep learning has also shown its power in clock-tree synthesis. GAN-CTS [15] introduces a multi-branch regression model that predicts power, wire length and clock skew. The framework also guides parameter settings of clock-tree synthesis that results in better performance.

Modern placement engines carry out their jobs in an optimization manner with heuristic fine-tuning [16–19]. Dealing with millions of cells has made direct machine learning solutions challenging. However, there are still attempts to bring deep learning tools into the flow that can benefit the physical implementation stage. Goldie *et al.* [20] proposes a deep reinforcement learning solution for floor planning. This work conducts macro block placement in a sequential scheme, where each step is completed according to the trained reinforcement learning policy. Graph neural network (GNN) is employed for meta data embedding and a trained net composed of multiple fully-connected layers is designed for reward prediction. Recently, Liu *et al.* [21] embeds explicit routability estimation into the global placement flow by regularizing cell placement position with a deep learning-based congestion estimator. In the global placement runtime, the gradient of congestion with respect to cell positions are backpropagated together with density and wirelength gradients.

Lithography hotspot detection and mask optimization are the earliest EDA tasks that benefit from the fast development of deep learning techniques. A major reason is the image nature of layout designs. Neural network architectures [22–27], training algorithms [24, 28, 29], and layout representations [28, 30] are deeply studied.

Shin *et al.* [27] and Yang *et al.* [22] are the two pioneers who employ deep neural networks for hotspot detection with image-based layout representation. Moreover, Yang *et al.* [28] investigates the frequency domain layout representation that makes the framework be storage-efficient. To apply the advanced hotspot detection technology on full chip layout, Chen *et al.* [10] designs a new faster region-based hotspot detection framework. Furthermore, Geng *et al.* [24] leverages the deep layout metric learning model to learn the representative layout feature embeddings seamlessly combining with hotspot detection tasks. Other architecture studies include binary neural networks [23] and ResNet [30]. Additionally, hotspot detection-aware training algorithms are developed, including label penalty [28] and AUC optimization [29].

Regarding mask optimization tasks, GAN-OPC [25] tries to fit conditional GAN for mask image generation. These GAN generated mask images are proved to be a better initialization for legacy OPC engine that offers faster convergence and final mask quality. DAMO [26] even builds up high performance DCGAN model for both mask generation and lithography modeling which outperforms an state-of-the-art industrial tool in terms of the VIA mask optimization.

2.2 Graph Neural Networks in EDA

As one of the most common and representative irregular data representations in EDA, graph has been well-studied in the modern design flow. Recently, many works have probed graph learning methods by modeling the target objects as graphs. These target objects mainly include: 1) *Netlist.* A netlist can be easily transformed into an undirected graph, where each cell corresponds to one node. Mirhoseini *et al.* [3] models the netlist as the graph and uses deep

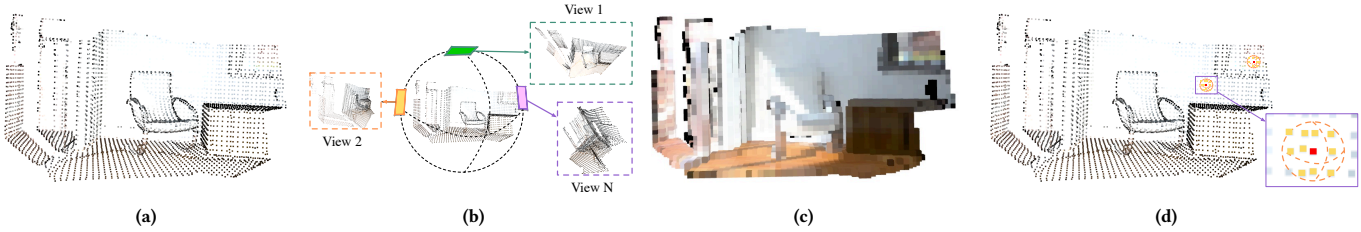


Figure 3: Three major kinds of methods for cloud embedding: (a) The original point cloud; (b) View-based methods; (c) Volumetric-based methods; (d) Point-based methods;

reinforcement learning (DRL) to pose placement, in which a GNN is adopted as the backbone to extract the graph embedding and node embeddings. Similarly, Wang *et al.* [7] obtains the graph embedding of corresponding netlist by GNN, the embedding is further passed to the DRL agent. Besides the application in DRL, the graph embedding can be directly used in some downstream tasks such as classification (timing model selection [6], test point insertion [2]), segmentation (3D partition [4]), and regression (net parasitics and device parameters predictions [5]). 2) *Layout.* Graph sometimes can be also a natural representation of layout. For example, the layout decomposition problem can be translated into a variation of graph coloring problem [31]. Following this kind of transformation, Li *et al.* [8] decomposes the layout with the help of graph embedding, which is used to select algorithm, construct library, and match graph.

3 Background of Point Clouds

Point cloud, as its name suggests, is the set of data points in space. Each point in a point cloud contains its location information, for example, a set of x, y and z coordinates if it is in a 3D space. The study of point clouds becomes increasingly important mainly because of the rapid development of 3D acquisition technologies. The emergence of affordable and available sensors brings about various types of data, in which point cloud is one of the most essential representations since it directly preserves the original geometric information without any discretization. Due to the increasing importance of point clouds and the presence of massive data, a great number of works [32, 33, 35, 40–47] are developed to explore the possibility of deep learning on point clouds. In this section, we first introduce previous deep learning based methods for point clouds, then we discuss challenges in EDA applications raised by some special characteristics of point clouds and how these issues are addressed by previous works and potential solutions.

3.1 Point Cloud Learning with Neural Networks

Guo *et al.* [48] categorizes deep learning based methods for point clouds into three major types: multi-view based methods, volumetric-based methods, and point-based methods. We give a simple illustration of these three methods in Figure 3.

3.1.1 Multi-view based Methods. Multi-view based methods [41–44] are designed for 3D point cloud. These methods first transform a 3D point cloud into multiple views through projection, and extract view-wise features. Finally, extracted features are fused together to generate a cloud embedding. Among these works, the major discrepancy locates in the fusion of multiple view-wise features, which is also the key challenge. For example, MVCNN [41] uses a straightforward max pooling operator to aggregate features; Yang et al. [43] fuse these features based on a relation network, which includes inter-relationships among regions and views. GVCNN [44] proposes a hierarchical view-group-shape architecture to obtain the cloud embedding from the view level, the group level, and the shape level.

3.1.2 Volumetric-based Methods. Volumetric-based methods also use the idea of transformation to solve the irregularity of regular data. Instead of views, these methods voxelize a point cloud into regular grids, and then a conventional CNN is compatible with the volumetric data for feature extraction. VoxNet [45], one of the pioneer works using a volumetric data, uses a volumetric occupancy grid representation and fed it into a 3D Convolutional Neural Network (3D CNN).

3.1.3 Point-based Methods. Compared to view-based methods and volumetric-based methods, which generate unavoidable information loss, point-based methods skip any preprocess techniques such as voxelization and projection, and directly handle with raw point clouds. Typical point-based methods usually include three procedures to obtain the embedding: *Sampling*, *Grouping* and *Encoding*. *Sampling* is to select a set of centroids from the original point cloud to reduce the memory cost. *Grouping* is to select a set of neighbors (also called agglomerates) for each centroid, which represents a local information and works like the local region constrained by a convolution kernel in the original convolution. *Encoding* is to encode the new centroid feature using the original one and the local feature aggregated from the neighbors of the centroid. In *Sampling* phase, widely used sample rules include *Farthest Point Sampling* (FPS) [32, 34, 36, 40], random sampling [35], Poisson disk sampling [39, 49], and VoxelGrid sampling [37, 50]. In *Grouping* phase, ball query grouping [33, 36, 49, 51] and k nearest neighbors (KNN) [35, 40, 52, 53] are the two dominant methods. As for methods used in *Encoding* phase, previous survey [48] lists three main types: MPL-based, convolution-based, and graph-based methods.

MPL-based methods use Multi-Layer Perceptrons (MLPs) as the backbone to extract hidden features. Among all these methods, PointNet [32] is the pioneering work which simply calculates point-wise features independently, causing a loss of neighborhood information. Following this way, many efforts have been made to further increase its representation power. For example, PointNet++ [33], the extension work of PointNet, captures neighborhood information by a hierarchical model on the basis of MLP. Besides solely using a MLP module, some works use attention technique to explore the

Table 1: Summary of existing point-based methods which follow a sampling-grouping-encoding scheme

Methods		*Sampling*	*Grouping*	*Encoding*
MLP-based methods	PointNet [32]	-	-	$v'_{ic} = \sigma(\boldsymbol{\theta}_c \boldsymbol{v}_i)$
	PointNet++ [33]	FPS	Ball query	$v'_{ic} = \max_{j \in E_i} \sigma(\boldsymbol{\theta}_c \boldsymbol{v}_j)$
	Yang et al. [34]	FPS/GSS	-	$v'_{ic} = \sigma(\boldsymbol{\theta}_c \boldsymbol{v}_i)$
Convolution-based methods	PointCNN [35]	Random/FPS	KNN	$\boldsymbol{v}'_i = Conv(X \times \boldsymbol{\theta}(\boldsymbol{v}_i - \boldsymbol{v}_j))$
	RS-Conv [36]	FPS	Ball query	$v'_{ic} = \sigma(\frac{1}{\lvert E_i \rvert} \sum_{j \in E_i} \boldsymbol{v}_i \times MLP(CONCAT(\boldsymbol{v}_i - \boldsymbol{v}_j, \boldsymbol{v}_i, \boldsymbol{v}_j)))$,
Graph-based methods	ECC [37]	VoxelGrid	KNN	$v'_{ic} = \frac{1}{\lvert E_i \rvert} \sum_{j \in E_i} F(\boldsymbol{v}_j)$,
	FoldingNet [38]	Random	KNN	$v'_{ic} = \boldsymbol{\theta}_c \cdot \max_{j \in E_i} \sigma(\boldsymbol{v}_j)$,
	KCNet [39]	Poisson disk	KNN	$v'_{ic} = \max_{j \in E_i} (\boldsymbol{\theta}_c \cdot \boldsymbol{v}_j)$,
	DGCNN [40]	-	KNN	$v'_{ic} = \max_{j \in E_i} \sigma(\boldsymbol{\theta}_c \cdot CONCAT(\boldsymbol{v}_i - \boldsymbol{v}_j, \boldsymbol{v}_i))$,

relational information, i.e., relations between neighbors and centroid [34], and between local structures [54].

Convolution-based methods process the centroid along with the neighborhood by a continuous [36, 55] or discrete [35, 56] convolutional kernel, i.e., a weighted sum over a given subset related to the centroid and the neighborhood. Among all convolution-based methods, PointCNN [35] is the most representative one. It does not use a symmetric function to keep the order invariance property. Instead, the relative coordinates are adopted to obtain a transformation matrix X, which transforms the relative coordinates into a latent canonical form and thus achieves the order invariance. Then the transformed input is processed by a convolution operation.

Graph-based methods treat each point in the point cloud as the vertex in the graph and connects each centroid with its selected neighbors. In this way, the original point cloud is viewed and processed as a graph, in which GNNs can be utilized. Similar with GNNs, graph-based methods also develop two different directions for feature learning: spatial domain and spectral domain. Methods in spatial domain obtain point embeddings following a convolution philosophy, i.e., recursively updating node features by aggregating information from neighbors repeatedly. The pioneer work [37] simply updates node feature by calculating the average value of all neighbor features processed by a filter-generating network F, e.g. MLP. Another work [38] designs an auto-encoder, in which the graph-based encoder replaces the average term as a max-pooling operation. Evolved from the static graph, DGCNN [40] proposes a dynamic graph construction method which updates the neighbors after each layer of the network. Methods in spectral domain implements the *Encoding* from a spectra perspective. For example, RGCNN [57] defines the convolution over graph by Chebyshev polynomial approximation and Wang *et al.* [58] update node features by standard unparametrized Fourier kernels.

For a clear comparison, we list most state-of-the-art point-based methods in Table 1. Some methods are not covered in the table since they do not follow a typical sampling-grouping-encoding scheme. The method shown in the table may only represent a part of the method. For example, FoldingNet [38] described in Table 1 solely stands for the encoder in the work.

3.2 Challenges in EDA applications

Deep learning has attracted more and more attention in kinds of domains. Without exception, deep learning also motivated a data-driven approach to learning features on point clouds. However, it is not an easy task to adopt deep learning on point clouds caused by several intrinsic characteristics of point clouds. We summarize these challenges as follows:

Order invariance. Given a point cloud, any permutation on the order of points has no influence on the embedding and the final results. However, most standard neural network models such as Convolutional neural network (CNN) and Recurrent neural network (RNN) are sensitive to the order permutation. For example, the permutation on the input sequence affects the inference of RNN inevitably. Therefore, any operation regarding to the point clouds should be order invariant. In the multi-view based methods and volumetric-based methods, order invariance is kept obviously by transformation, i.e., projection and voxelization. Among various point-based methods, most of them achieve order invariance by some symmetric functions like max pooling or summation. Some convolution-based methods design a special trainable network to maintain the order invariance such as PointCNN [35] described before.

Irregularity. It is straightforward to feed regular data representations, such as sequences, images, and videos, into a deep learning model. In contract, finding or designing a suitable network architecture for irregular data is non-trivial since it usually varies in size and distributes in the non-Euclidean space. As introduced in Section 3.1, there are two adopted solutions to tackle the irregularity. Both multi-view based methods and volumetric-based methods transform the irregular point cloud into a regular grid-like data such as image or voxel. On the contrary, point-based methods directly work on points and propose networks specifically for irregular data like GNNs.

Sparsity. In addition to the compatibility problem, the *sparsity* nature of point clouds elicits the concern about the efficiency. That is, the concern about how to extract the high-density information from a sparse point cloud, and how to process massive sparse data under the computation resource and time constraints. The issue is strikingly important especially for volumetric-based methods: these methods usually require a prohibitive memory and computation resources for storing volumetric data considering that point clouds are usually scattered in the space sparsely. Therefore, more advanced and compact structures are needed to reduce the computational and memory costs of these methods by exploiting the sparsity. OctNet [46] proposes a set of unbalanced octrees and each leaf node stores a pooled feature. These unbalanced octrees partition the space hierarchically and are encoded by a bit string representation so that the model focused on dense regions efficiently. The structure of octree

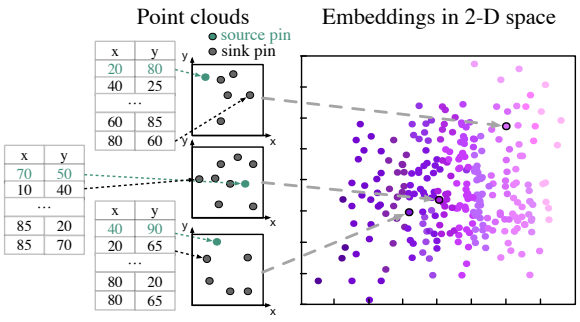


Figure 4: Cloud embeddings for tree construction, where point clouds are transformed into unified 2-D Euclidean space.

is encoded using a bit string representation, and the feature vector of each voxel is indexed by simple arithmetic.

Dimension. Apart from these natures that impede the applications in deep learning, there are also some additional challenges particularly in EDA applications. One of the most crucial concerns is about dimension. Previous works usually handle 3D objects in real-world while 2D scenario is the most principal one in the EDA field. Figure 4 gives a demonstration from 2D point clouds for the tree construction to their embeddings. Therefore, methods that are flexible in dimension should receive more attentions while others, e.g. view-based methods, may not be appropriate for EDA problems. Luckily, most point-based methods demonstrate a strong adaptability in dimension which can be applied in 2D point clouds directly or through some simple dimensionality reduction transformation. For example, a 3D ball query corresponds to a 2D circle query in most EDA problems.

4 Case Study 1: Routing Tree Construction

In this section, we discuss the routing tree construction problem as a case study, whose input can be formalized as a special point cloud directly. The target of the routing tree is to minimize both wirelength (WL) and pathlength (PL). Among all construction methods, PD-II [59] and SALT [60] show the best performance in terms of the trade-off between WL and PL. However, given a specific net, it is still hard to say which one, PD-II or SALT, is better. Also, it is non-trivial to select the parameter that is used to control the balance between WL and PL (α in PD-II, ϵ in SALT). Although the routing tree is constructed by a set of terminals, i.e., a 2-D point cloud. the input terminals possess some special properties compared with a general point cloud. We start from the analysis of these properties and propose a deep learning-based model, TreeNet, to obtain the cloud embedding specifically for the tree construction based on these properties. The obtained cloud embedding is then used to help select the best routing tree construction method and predict the best parameter for the selected method.

4.1 Routing Tree

The input of the routing tree $\boldsymbol{V} = \{v_0, \boldsymbol{V}_s\}$ is composed of the source (v_0) and the set of sinks ($\boldsymbol{V}_s$). Constructed from $\boldsymbol{V}$, the routing tree $T = \{\boldsymbol{V}', \boldsymbol{E}'\}$ is a spanning/Steiner tree with v_0 as the root. A Steiner tree inserts additional points during the construction, i.e., $\boldsymbol{V}' \supseteq \boldsymbol{V}$, and the newly inserted points are called steiner points. The objective of the routing tree is to minimize both WL and PL. The WL metric is called the lightness or normalized WL, which is computed by the WL ratio with that of minimum spanning tree (MST), i.e., $lightness = \frac{w(T)}{w(MST(G))}$, where $w(\cdot)$ is the total weight and G is the connected weighted routing graph. The PL metric is controversial and there are two widely used metrics. The first one is called the shallowness [61], which is computed by the maximal PL ratio with the shortest-path tree (SPT) among all vertices, i.e., $shallowness = \max\{\frac{d_T(v_0,v)}{d_G(v_0,v)} | v \in V_s\}$. The second one is called the normalized path length [59], which is computed by the total PL normalized by the total shortest-path distance, i.e., $normPL = \frac{\sum_{v\in\boldsymbol{V}} d_T(r,v)}{\sum_{v\in\boldsymbol{V}} d_G(r,v)}$. Note that $d(\cdot)$ mentioned above denotes as the Manhattan distance.

4.2 Property Analysis

Although we can use previous methods to obtain the cloud embedding directly, the previous ones may be detrimental to the representation power of final obtained embedding, especially for those point clouds which own some special properties. The point clouds for the routing tree construction are one of those special point clouds. These properties followed by corresponding analysis are described as follows:

1) *Sampling*. Unlike traditional point cloud which may have thousands of points to represent a surface of one entity, the size of the point cloud for the tree construction is usually small (e.g., up to 60). Also, every point for the tree construction is indispensable as the node in the corresponding tree. Therefore, the widely-used down-sampling tricks such as random sampling and farthest point sampling are not applicable for our application.

2) *Root*. Given a point cloud for the tree construction, there is a critical point called the root, corresponding to the source in the constructed routing tree. Previous methods treat all points equally, which may be a problem. Let's consider a pair of point clouds for the tree construction shown in Figure 5, whose point locations are totally the same and only the root is assigned differently. In this case, a typical method may regard the two point clouds as a similar pair while they actually represent completely different trees.

3) *Grouping methods*. The dominant grouping methods in previous works are mainly KNN and ball query. However, both grouping methods fail to capture the structure of routing tree in some cases such as the example shown in Figure 6. On the contrary, the graph G_{bbox} (Figure 6(c)) whose nodes are connected with their *bbox-neighbors* shows an exact match with the routing tree (a subset relation). Here, we call the node u_j as the *bbox-neighbor* of u_i if there is no other node in the smallest bounding box containing u_j and u_i. Actually, it is not difficult to show that a minimum spanning tree is always the subgraph of G_{bbox}.

4.3 Routing Tree Construction based on Point Cloud Embedding

Based on all properties discussed above, we first develop a specialized convolution-like operation, TreeConv, to obtain the cloud embedding. TreeConv follows a classical procedure with previous methods to generate the embedding, i.e., *Sampling*, *Grouping* and *Encoding*. Differently, each phase is specifically designed considering those special properties mentioned above. In summary, we 1)

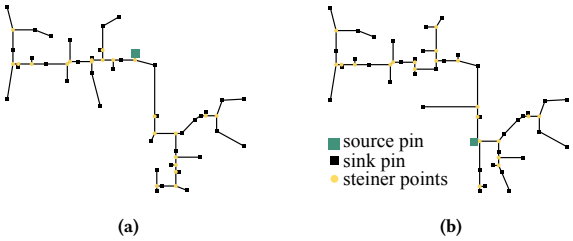


Figure 5: Examples of the routing trees with the same node distribution but different root (highlighted by red).

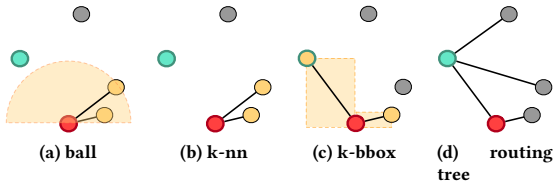


Figure 6: Comparison among (a) ball query, (b) k-nn, and (c) k-bbox grouping methods ($k = 2$ in this example). The orange regions represent the query ball in (a) and bounding boxes in (c). The centroid is highlighted by red and the root is by green.

discard any kind of down-sampling; 2) design a grouping rule based on G_{bbox}; 3) utilize the critical root information in *Encoding* phase. 4) obtain the cloud embedding by two permutation-invariant operations, i.e., max pooling and average pooling. 5) use a root-sensitive normalization. After we obtain the cloud embedding by the stack of TreeConv, we can cast the algorithm selection and the parameter prediction problem into classification and regression problem respectively, and solve them by cloud embedding directly. For more detais, we refer readers to [62] for a comprehensive understanding of the whole workflow.

We compare TreeNet with other state-of-the-art models [32, 33, 35, 40] for the algorithm selection task. The result for the algorithm selection is shown in Table 2, where "Recall*" is the fraction of the total amount of positive instances that were also predicted as positive with the confidence larger than the bar b. Thus, the value in "Recall*" directly indicates the effectiveness of corresponding method. We compare four state-of-the-art models with our TreeNet and three variations: 1) Remove the root-sensitive normalization and use the original normalization; 2) Remove the root-related global information in *Encoding* phase; 3) Use k-nn grouping method instead of k-bbox.

We have the following observations: (1) Our TreeNet outperforms other state-of-the-art models on all three metrics. DGCNN [40] is the most closest one with a similar accuracy and precision. However, the low recall results in a terrible efficiency loss and makes it far inferior to TreeNet. This makes sense since DGCNN use the same sampling strategy and a similar encoding method, which also demonstrates the correctness of our analysis on the *sampling* and *encoding* phases. (2) Our original TreeNet is the best among all variations. All variations

Table 2: Algorithm selection results

Method	Accuracy	Precision	Recall*
PointNet [32]	54.13	53.95	1.91
PointNet++ [33]	81.31	82.50	2.65
PointCNN [35]	62.18	64.24	1.16
DGCNN [40]	92.24	94.62	11.84
TreeNet w.o. Nor	87.22	88.62	15.69
TreeNet w.o. global	92.40	94.63	25.53
TreeNet w. knn	92.58	94.79	26.76
TreeNet	**94.09**	**95.38**	**50.74**

harm the model performance, which validates our analysis based on the properties of the point cloud for the tree construction.

5 Case Study 2: Hotspot Detection

In learning based EDA methodologies, feature representation has always been a significant concept. Recently, there are various studies demonstrating that advanced feature representation could lead to superior performance as well as efficiency improvement [63, 64]. In this section, we explore the possibility of hotspot detection with point cloud embedding.

5.1 Point Cloud Based Layout Representation

Our transformation workflow from layout representation to point cloud can be summarized as follows: (1) *Extraction*: Coordinates for describing the patterns are extracted according to the designs. Here we extract vertexes from polygons as points. A point l is represented by the coordinate x, y and z. (2) *Augmentation*: Points are discretized into an evenly spaced grid in the x-y plane, a set of pillars P are created. The points in each pillar are augmented with x_c, y_c, z_c, x_p, and y_p, where the c subscript denotes the distance to the arithmetic center of all points in the pillar and the p subscript denotes the distance between points and pillar center. From the perspective of feature representation, using pillars as the descriptor to represent the patterns is equivalent to images and much more efficient.

5.2 Point Cloud Hotspot Detection Model

In this work, we extend the point-based feature extractor based on PointNet++ [33] to build up a two-stage point-based hotspot detector, which directly generate hotspot box proposals and detection results from raw point clouds.

Our proposed point-based hotspot detection framework is illustrated in Figure 7. For detecting hotspot from irregular points, the detector consists of the hotspot box proposal generation stage and the bounding box refinement stage. In this part, we will discuss each stage with details.

(1) *Hotspot box proposal generation*: To fully utilize point-wise features from the point cloud, PointNet++ [33] with multi-scale grouping is applied to serve as the backbone of our model. As the extension of PointNet [32], PointNet++ applies PointNet recursively on a nested portioning of input point cloud. The *Farthest Points Sampling* (FPS) algorithm is used to select centroids, and ball query is used to extract local features from a small neighborhood thus further grouping into larger units to produce higher level features. Equipped with the point-wise features encoded by the PointNet++ backbone, we add one segmentation head for predicting foreground points information and one box regression head for generating hotspot

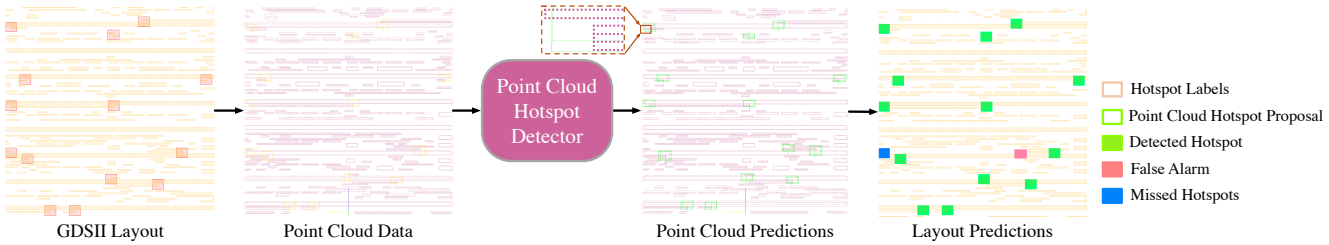


Figure 7: Overall flow of point cloud hotspot detection model.

Table 3: Comparison with State-of-the-art Hotspot Detectors

Bench	Faster R-CNN [65]			TCAD'18 [11]			TCAD'20 [10]			PCloud-HSD		
	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)
Case2	1.8	3	1.0	77.78	48	60.0	93.02	17	2.0	83.1	36	1.6
Case3	57.1	74	11.0	91.20	263	265.0	94.5	34	10.0	88.4	89	8.2
Case4	6.9	69	8.0	100	511	428.0	100	201	6.0	100	294	5.5
Average	21.9	48.7	6.67	89.66	274	251	95.8	84	6	90.5	139.6	5.1
Ratio	0.23	0.58	1.11	0.94	3.26	41.83	**1**	**1**	1	0.95	1.66	**0.85**

proposals. After the point segmentation and regression, we design point-based hotspot non-maximum suppression strategy to remove the redundant hotspot proposals. The algorithm which based on the oriented IoU from bird-eye-view is similar to the h-NMS strategy in [10]. We use 0.80 for IoU threshold and after hotspot non-maximum suppression we keep top 200 hotspot proposals for the next refinement stage.

(2) *Hotspot box refinement*: After the first stage, we get a rough segmentation results and hotspot box proposals, then we combine the local spatial features and their global semantic information through concatenation and several fully-connected layers. The aggregated embedding is further used to refine hotspot proposals and predict confidence for each proposal. We adopt the regression losses for the proposal refinement, where a ground-truth box is assigned to the hotspot proposals for learning box refinement.

5.3 Preliminary Results

We implement our point-based hotspot detection framework with Pytorch [66] in Python, and test it on a platform with an Nvidia GTX Titan GPU. Our point-based hotspot detection model is evaluated on ICCAD CAD Contest 2016 Benchmarks [67], which contains four designs are shrunk to match EUV metal layer design rules. Ground truth hotspot locations are label according to the results of industrial 7nm metal layer EUV lithography simulation under a given process window. As mentioned in Section 5.1, before fed into our model, the GDSII format layout will be transferred into point based datasets. The hotspot ground-truth label will be transformed to the ground-truth bounding box for detection. For each point-cloud scene in the training set, we subsample 32,768 points from each scene as the inputs, which came from $2560 \times 2560\ nm^2$ layout. For scenes with the number of points fewer than 32,768, we randomly repeat the points to obtain 32,768 points. Four feature propagation layers are then used to obtain the per-point feature vectors for segmentation and proposal generation. For the box proposal refinement sub-network, we randomly sample 512 points from the pooled region of each proposal as the input of the refinement sub-network.

The preliminary results of our point-based hotspot detector are detailed listed in Table 3. We use three benchmarks from [67] which are listed in Table 3 Column "Bench". Columns "Accu", "FA", "Time" stand for accuracy, false alarm count and runtime respectively. Column "TCAD'18" shows the results of [11], and Columns "Faster R-CNN" and "TCAD'20" come from [10]. Column "PCloud-HSD" is our proposed point cloud based hotspot detector. The results demonstrate that our framework can save 15% runtime comparing with TCAD'20 [10].

6 Conclusion

In this paper, we discussed possible applications of point clouds in EDA fields. We first introduced deep learning techniques used in EDA, followed by the backgrounds of point clouds including some challenges of point clouds applications and previous deep learning methods for point clouds. Two case studies on routing tree construction and hotspot detection were covered and demonstrated the promise of point clouds in the EDA domain. However, we only consider the two cases as prompts for an open thread of point clouds in EDA, e.g. a reinforcement learning method using point clouds model as the backbone.

Acknowledgment

This work is partially supported by National Key R&D Program of China 2020YFA0711900, 2020YFA0711903, and The Research Grants Council of Hong Kong SAR (No. CUHK14209420).

References

[1] Y. Zhang, H. Ren, and B. Khailany, "GRANNITE: Graph neural network inference for transferable power estimation," in *Proc. DAC*, 2020, pp. 1–6.
[2] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, "High performance graph convolutional networks with applications in testability analysis," in *Proc. DAC*, 2019, pp. 1–6.
[3] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae *et al.*, "Chip placement with deep reinforcement learning," *arXiv preprint*, 2020.

[4] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim, "TP-GNN: a graph neural network framework for tier partitioning in monolithic 3D ics," in *Proc. DAC*. IEEE, 2020, pp. 1–6.

[5] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *Proc. DAC*, 2020, pp. 1–6.

[6] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu, "Understanding graphs in EDA: From shallow to deep learning." in *Proc. ISPD*, 2020, pp. 119–126.

[7] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. DAC*, 2020, pp. 1–6.

[8] W. Li, J. Xia, Y. Ma, J. Li, Y. Liny, and B. Yu, "Adaptive layout decomposition with graph embedding neural networks," in *Proc. DAC*, 2020, pp. 1–6.

[9] U. Brenner, A. Hermann, N. Hoppmann, and P. Ochsendorf, "Bonnplace: A self-stabilizing placement framework," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 9–16.

[10] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *Proc. DAC*, 2019, pp. 146:1–146:6.

[11] H. Yang, J. Su, Y. Zou, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," in *Proc. DAC*, 2017, pp. 62:1–62:6.

[12] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and N. Corporation, "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. ICCAD*, 2018.

[13] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu, "DRC hotspot prediction at sub-10nm process nodes using customized convolutional network," in *Proc. ISPD*, 2020.

[14] J. Chen, J. Kuang, G. Zhao, D. J.-H. Huang, and E. F. Young, "PROS: A plug-in for routability optimization applied in the state-of-the-art commercial eda tool using deep learning," in *Proc. ICCAD*. IEEE, 2020, pp. 1–8.

[15] Y.-C. Lu, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim, "GAN-CTS: A generative adversarial framework for clock tree prediction and optimization," in *Proc. ICCAD*. IEEE, 2019, pp. 1–8.

[16] C. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, vol. 38, no. 9, pp. 1717–1730, 2019.

[17] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert, and D. Z. Pan, "MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," *IEEE TCAD*, vol. 37, no. 6, pp. 1237–1250, 2018.

[18] H. Li, W.-K. Chow, G. Chen, E. F. Young, and B. Yu, "Routability-driven and fence-aware legalization for mixed-cell-height circuits," in *Proc. DAC*, 2018, pp. 1–6.

[19] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *Proc. DAC*, 2019.

[20] A. Goldie and A. Mirhoseini, "Placement optimization with deep reinforcement learning," in *Proc. ISPD*, 2020, pp. 3–7.

[21] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Faster optimal single-row placement with fixed ordering," in *Proc. DATE*, 2021.

[22] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: a deep learning approach," *JM3*, vol. 16, no. 3, p. 033504, 2017.

[23] Y. Jiang, F. Yang, H. Zhu, B. Yu, D. Zhou, and X. Zeng, "Efficient layout hotspot detection via binarized residual neural network," in *Proc. DAC*, 2019, pp. 147:1–147:6.

[24] H. Geng, H. Yang, L. Zhang, J. Miao, F. Yang, X. Zeng, and B. Yu, "Hotspot detection via attention-based deep layout metric learning," in *Proc. ICCAD*, 2020.

[25] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," *IEEE TCAD*, 2019.

[26] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "Damo: deep agile mask optimization for full chip scale," in *Proc. ICCAD*, 2020, pp. 1–9.

[27] M. Shin and J.-H. Lee, "Accurate lithography hotspot detection using deep convolutional neural networks," *JM3*, vol. 15, no. 4, p. 043507, 2016.

[28] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," *IEEE TCAD*, vol. 38, no. 6, pp. 1175–1187, 2019.

[29] W. Ye, Y. Lin, M. Li, Q. Liu, and D. Z. Pan, "LithoROC: lithography hotspot detection with explicit ROC optimization," in *Proc. ASPDAC*, 2019, pp. 292–298.

[30] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Detecting multi-layer layout hotspots with adaptive squish patterns," in *Proc. ASPDAC*, 2019, pp. 299–304.

[31] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE TCAD*, vol. 34, no. 3, pp. 433–446, 2015.

[32] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. CVPR*, 2017, pp. 652–660.

[33] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. NIPS*, 2017, pp. 5099–5108.

[34] J. Yang, Q. Zhang, B. Ni, L. Li, J. Liu, M. Zhou, and Q. Tian, "Modeling point clouds with self-attention and gumbel subset sampling," in *Proc. CVPR*, 2019, pp. 3323–3332.

[35] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on x-transformed points," in *Proc. NIPS*, 2018, pp. 820–830.

[36] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *Proc. CVPR*, 2019, pp. 8895–8904.

[37] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. CVPR*, 2017, pp. 3693–3702.

[38] Y. Yang, C. Feng, Y. Shen, and D. Tian, "Foldingnet: Point cloud auto-encoder via deep grid deformation," in *Proc. CVPR*, 2018, pp. 206–215.

[39] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," in *Proc. CVPR*, 2018, pp. 4548–4557.

[40] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM TOG*, vol. 38, no. 5, pp. 1–12, 2019.

[41] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. CVPR*, 2015, pp. 945–953.

[42] X. Wei, R. Yu, and J. Sun, "View-GCN: View-based graph convolutional network for 3D shape analysis," in *Proc. CVPR*, 2020, pp. 1850–1859.

[43] Z. Yang and L. Wang, "Learning relationships for multi-view 3D object recognition," in *Proc. ICCV*, 2019, pp. 7505–7514.

[44] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, "GVCNN: Group-view convolutional neural networks for 3D shape recognition," in *Proc. CVPR*, 2018, pp. 264–272.

[45] D. Maturana and S. Scherer, "Voxnet: A 3D convolutional neural network for real-time object recognition," in *Proc. IROS*. IEEE, 2015, pp. 922–928.

[46] G. Riegler, A. Osman Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," in *Proc. CVPR*, 2017, pp. 3577–3586.

[47] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3D shape analysis," *ACM TOG*, vol. 36, no. 4, pp. 1–11, 2017.

[48] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3D point clouds: A survey," *IEEE TPAMI*, 2020.

[49] P. Hermosilla, T. Ritschel, P.-P. Vázquez, À. Vinacua, and T. Ropinski, "Monte carlo convolution for learning on non-uniformly sampled point clouds," *ACM TOG*, vol. 37, no. 6, pp. 1–12, 2018.

[50] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (pcl)," in *Proc. ICRA*. IEEE, 2011, pp. 1–4.

[51] H. Lei, N. Akhtar, and A. Mian, "Octree guided CNN with spherical kernels for 3D point clouds," in *Proc. CVPR*, 2019, pp. 9631–9640.

[52] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "SpiderCNN: Deep learning on point sets with parameterized convolutional filters," in *Proc. ECCV*, 2018, pp. 87–102.

[53] F. Groh, P. Wieschollek, and H. P. Lensch, "Flex-Convolution," in *Proc. ACCV*. Springer, 2018, pp. 105–122.

[54] Y. Duan, Y. Zheng, J. Lu, J. Zhou, and Q. Tian, "Structural relational reasoning of point clouds," in *Proc. CVPR*, 2019, pp. 949–958.

[55] A. Boulch, "ConvPoint: Continuous convolutions for point cloud processing," *Computers & Graphics*, 2020.

[56] B.-S. Hua, M.-K. Tran, and S.-K. Yeung, "Pointwise convolutional neural networks," in *Proc. CVPR*, 2018, pp. 984–993.

[57] G. Te, W. Hu, A. Zheng, and Z. Guo, "Rgcnn: Regularized graph cnn for point cloud segmentation," in *acmmm*, 2018, pp. 746–754.

[58] C. Wang, B. Samari, and K. Siddiqi, "Local spectral graph convolution for point set feature learning," in *eccv*, 2018, pp. 52–66.

[59] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu, and S. Venkatesh, "Prim-Dijkstra revisited: Achieving superior timing-driven routing trees," in *Proc. ISPD*, 2018, pp. 10–17.

[60] G. Chen and E. F. Young, "SALT: provably good routing topology by a novel steiner shallow-light tree algorithm," *IEEE TCAD*, 2019.

[61] G. Chen, P. Tu, and E. F. Young, "SALT: provably good routing topology by a novel Steiner shallow-light tree algorithm," in *Proc. ICCAD*, 2017, pp. 569–576.

[62] W. Li, Y. Qu, G. Chen, Y. Ma, and B. Yu, "TreeNet: Deep point cloud embedding for routing tree construction," in *Proc. ASPDAC*, 2021.

[63] H. Zhang, B. Yu, and E. F. Y. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *Proc. ICCAD*, 2016, pp. 47:1–47:8.

[64] F. Yang, S. Sinha, C. C. Chiang, X. Zeng, and D. Zhou, "Improved tangent space based distance metric for lithographic hotspot classification," *IEEE TCAD*, vol. 36, no. 9, pp. 1545–1556, 2017.

[65] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. NIPS*, 2015, pp. 91–99.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. NIPS*, 2019, pp. 8024–8035.

[67] R. O. Topaloglu, "ICCAD-2016 CAD contest in pattern classification for integrated circuit design space analysis and benchmark suite," in *Proc. ICCAD*, 2016, pp. 41:1–41:4.