Journal of Intelligent Information Systems
https://doi.org/10.1007/s10844-025-00921-z

RESEARCH

PolyCard: A learned cardinality estimator for intersection queries on spatial polygons

Yuchen Ji[1] · Daichi Amagata[1] · Yuya Sasaki[1] · Takahiro Hara[1]

Received: 17 September 2024 / Revised: 7 January 2025 / Accepted: 8 January 2025

Abstract
How can we estimate the result size for a given query on complex spatial objects like polygons? Estimating a query's result size, also known as the cardinality estimation, plays a significant role in query scheduling and optimization. Accurate and fast cardinality estimation substantially improves query efficiency. Existing compatible solutions, mainly histogram-based, deal with polygons as their minimal bounding rectangles for easier processing, which leads to inaccurate estimation. To address this issue, we present PolyCard, a learned cardinality estimator for intersection queries on spatial polygons. We successfully apply learning techniques to spatial polygons with variable sizes. PolyCard has the following properties. (i) Accurate: PolyCard improves 30% accuracy compared with existing solutions, (ii) Fast: PolyCard takes only 4 microseconds for an estimation, and (iii) Stable: PolyCard is robust against datasets and queries of different cardinalities. Our experiments on four real-world datasets of millions of polygons demonstrate the efficiency and effectiveness of PolyCard.

Keywords Cardinality estimation · Query optimization · Spatial polygon · Machine learning

1 Introduction

Spatial applications, such as mobile maps and geographical information systems (GIS), play a significant role in current society (Amagata & Hara, 2019, 2016, 2017; Hori et al., 2023; Pandey et al., 2021; Amagata et al., 2022; Aji et al., 2012; Hayashida et al., 2018; Nishio et al., 2022; Ji et al., 2024). These applications require many spatial objects like polygons

✉ Yuchen Ji
ji.yuchen@ist.osaka-u.ac.jp

Daichi Amagata
amagata.daichi@ist.osaka-u.ac.jp

Yuya Sasaki
sasaki@ist.osaka-u.ac.jp

Takahiro Hara
hara@ist.osaka-u.ac.jp

[1] Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan

and efficient techniques that process such spatial objects (Bouros & Mamoulis, 2019; Sasaki, 2021). Many spatial objects in real-world applications are represented as polygons. For example, in map applications, districts, rivers, and parks are represented as spatial polygons (or a set of polygons), and this article focuses on spatial polygons. One of the fundamental operations employed in these applications is *intersection query*, which, given a query and a set of spatial polygons, finds all spatial polygons intersecting with the query in the dataset. As shown in Fig. 1, this query supports (i) finding all districts crossed by a high-risk river during heavy rainfall[1] and (ii) looking for parks located within a specific region using a map application (Jacox & Samet, 2007; Mamoulis et al., 2004). These examples trivially clarify the importance of intersection queries.

Low response time for queries is an essential requirement for application users. Database management systems (DBMS) usually use a cost-based query optimizer to generate the most efficient query plan and to achieve low query latency. The query optimization heavily relies on good estimates of the cardinality of query results (Shekelyan et al., 2021; Ji et al., 2022; Shi et al., 2023). Therefore, to process intersection queries on spatial polygons efficiently, we need an estimator that estimates the cardinality of a given intersection query with high accuracy.

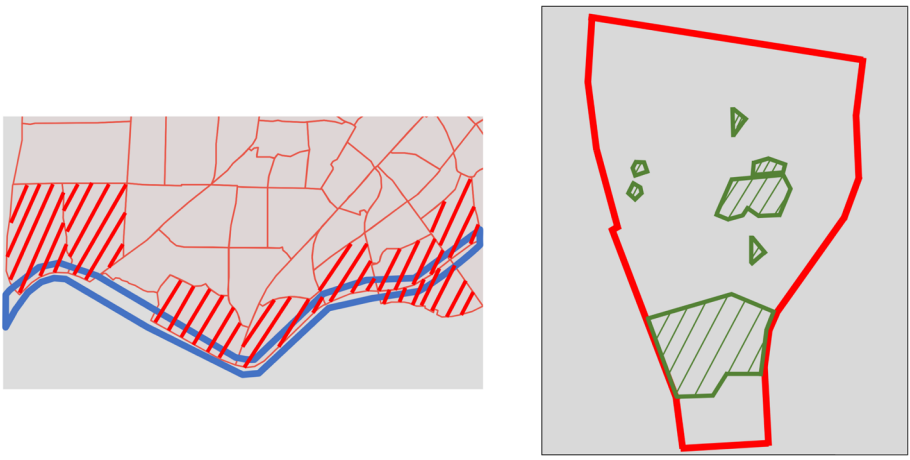
1.1 Motivation and challenge

Existing cardinality estimation methods for polygons typically rely on histograms built on the minimum bounding rectangles (MBRs) of polygons (Leis et al., 2015). They estimate the cardinality of a given query's result based on the statistics derived from the histograms. Unfortunately, these methods have two drawbacks: (1) They approximate polygons by their MBRs, which makes the estimation essentially inaccurate. (2) They cannot maintain stable performance for different datasets and queries. For example, the estimation time of histogram-based methods is highly dependent on the number of buckets accessed. Some methods designed to optimize spatial intersection searches/joins can also be used for cardinality estimation (Bouros & Mamoulis, 2019). RI (Georgiadis & Mamoulis, 2023) provides efficient approximations for polygons, particularly for the spatial intersection join cases. However, RI is too slow to estimate the cardinality of a single intersection query's result.

In recent years, learned cardinality estimation methods have been proposed to provide efficient estimations for relational database systems (Dutt et al., 2019; Yang et al., 2019; Wang et al., 2020, 2021b). They have shown great advantages compared with traditional histogram-based methods. Most of them adopt deep neural networks and train these networks in a supervised manner. However, they cannot be used for spatial polygons. Each polygon is represented by a set of two-dimensional coordinates. The number of coordinates is variable, e.g., from three to hundreds in real-world spatial datasets (Eldawy & Mokbel, 2015). The variable length input is not allowed for neural networks. The most common solution to this issue is padding all the input data with zeros (Hashemi, 2019). Considering the possible sizes of polygons, the zero padding greatly increases the computation cost for both training and estimation. Low latency is always required for the cardinality estimation problem because the cardinality estimation is frequently called in a database system.

Moreover, collecting high-quality training data is always challenging for learned cardinality estimators (Kipf et al., 2019; Kwon et al., 2022) because the skewed distribution of polygons in real-world polygon datasets makes obtaining sufficient training data difficult. One may come up with randomly generating query (training) polygons to address this issue.

[1] These districts are from https://geoshape.ex.nii.ac.jp/.



(a) (b)

Fig. 1 Examples of intersection queries. (a) Find districts in a city (red polygons) crossed by the bottom river (the blue polygon). The results are denoted by the red polygons with slashes. (b) List parks (green polygons with slashes) located in the specified district (the red polygon)

This is, however, not appropriate because the shape and the number of vertices of polygons are not fixed. Furthermore, the cardinality distribution of randomly generated queries has a long tail. The distribution with a long tail makes convergence of the training of the network difficult (Yosinski et al., 2015).

As can be seen above, existing techniques are not appropriate for cardinality estimation of intersection queries on polygons, despite the importance of efficiently supporting this task. Motivated by this fact, we devise an efficient and accurate cardinality estimator for intersection queries on polygons. Due to the potentially high accuracy, this article considers a machine learning (ML) approach. Recall that simply employing a ML approach is not feasible, so we overcome the following main challenges.

- Variable input size: Neural networks need to be compatible with input polygons of variable sizes.
- Training data generation: The high-quality and sufficient training data is desired for a learning method.
- Fast estimation: The estimation time should be at least competitive with light-weight methods, e.g., histogram-based ones, and be quite fast (e.g., microsec-order).

1.2 Contribution

In this article, we propose a learned cardinality estimation method for polygons, providing fast and accurate estimations for spatial intersection queries. We summarize our contributions below.

(i) PolyCard We propose PolyCard, the first learned cardinality estimator for intersection queries on polygons. The main idea behind PolyCard is to make polygons of variable sizes compatible with neural networks and provide substantial estimation performance derived from learning techniques. We propose an adaptive sampling method to transform polygon data to a fixed size, which considers the spatial distribution of the coordinates belonging to a

polygon. The differences between polygons are well preserved even after the transformation. The transformed polygons can be processed efficiently after normalization and sent into a specifically designed neural network. This technique yields the final estimation to be fast and accurate.

(ii) Training data generation PolyCard is a supervised estimator, so its performance depends on training data (i.e., training queries). We propose a training data generator to provide training data distributed over the data space and covering different cardinalities. The distribution of polygons is naturally skewed. The long tail distribution of true cardinalities corresponding to randomly generated training queries heavily burdens the training process. We guarantee an even cardinality distribution of training data by a sampling strategy.

(iii) Experiments We conduct experiments on four real-world datasets. Our results demonstrate that PolyCard outperforms state-of-the-art cardinality estimation methods and can maintain stable performances on different datasets. Figure 2 shows the overview of our results: PolygCard is very fast and has a better trade-off between speed and accuracy (q-error) than the competitors.

1.3 Roadmap

The rest of this article is organized as follows. We present the preliminary information in Section 2. We elaborate on PolyCard in Section 3. We report our experimental results in Section 4, and Section 5 concludes this article.

2 Preliminary

In this section, we define our problem and review related works. Table 1 gives the main symbols we use in this article.

2.1 Definition

Let D denote a set of polygons. A polygon is a typical spatial object type. We use P to denote a polygon, and it is defined as:

Definition 1 (POLYGON) P is represented by a list of vertices, i.e., $P = [p_1, p_2, ..., p_m]$. Each of its vertices p is represented by the two-dimensional coordinates, i.e., $p = (x, y)$.

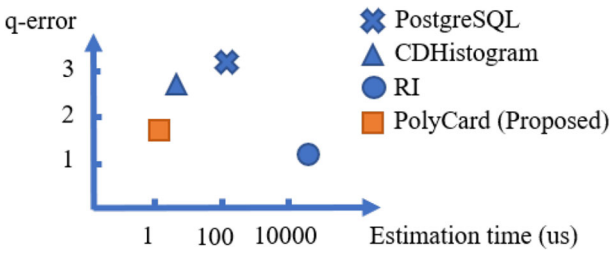


Fig. 2 Accuracy (represented by the 50th q-error) vs. estimation time on Sports dataset

Table 1 Overview of symbols frequently used in this article

Symbol	Description
D	Polygon dataset
P	Polygon in a dataset D
p	Vertex in a polygon P
Q	Intersection query
$card(Q)$	True cardinality of Q
$\widehat{card(Q)}$	Estimated cardinality of Q

There is an edge (or a line segment) between consecutive vertices p_i and p_{i+1} (p_m and p_1 are also consecutive vertices).

That is, a polygon is a two-dimensional shape closed by its edges. Notice that each polygon can have a different number of vertices.

This article considers intersection queries on polygons, and we define this type of query and its cardinality below. (Recall that Fig. 1 illustrates some examples of intersection queries.)

Definition 2 (Polygon Intersection) Polygon A is intersected with polygon B if one of the following cases is met:

- Any pair of edges $\langle e_{ai}, e_{bj} \rangle$ intersects, where e_{ai} is an edge of A and e_{bj} is an edge of B.
- A (B) is fully contained in B (A).

The polygon intersection is first tested using the MBRs of two polygons. If MBRs do not overlap, the two polygons do not intersect. Otherwise, a line segment intersection test is used to check if any pairs of line segments (edges) are intersected (Rigaux et al., 2001; Liu & Puri, 2020). If no pairs of line segments intersect, a point-in-polygon test can determine if one polygon is fully contained within the other (Heckbert, 2013).

Definition 3 (Cardinality of Intersection query) Given a dataset D and an intersection query Q represented by a query polygon q, the intersection query Q returns all polygons in D that intersect with q. The cardinality of Q is the output size of Q, i.e., the number of polygons in D that intersect with Q. We use $card(Q)$ to denote the cardinality of Q.

The notion of *selectivity* is close to cardinality and represented as $\frac{card(Q)}{N}$, where N is the size of the dataset.

It takes $O(m \log m)$ time to test if two polygons are intersected and $O(Nm \log m)$ time for an intersection query, where m is the number of vertices in a polygon and N is the number of polygons in D. We address the cardinality *estimation* problem to help further accelerate query processing in this article. We use $\widehat{card(Q)}$ to denote an estimated cardinality of Q. We aim at fast and accurate estimation.

2.2 Related works

Histogram-based cardinality estimation. Histogram-based cardinality estimation methods are the most widely used in relational database systems like PostgreSQL (Leis et al., 2015; Woltmann et al., 2021). These database systems build histograms on relational tables as

the statistics. Estimators for rectangle queries have also been developed for spatial cases (Derthick et al., 1999; Belussi et al., 2004).

CDHistogram addresses this problem by maintaining four sub-histograms corresponding to the four vertices of a rectangle (Jin et al., 2000). A bucket in a sub-histogram stores the number of corresponding vertices falling in the bucket. This design can avoid duplicate counts of rectangles spanning several buckets. Histogram-based methods can be used for our task by approximating polygons with their MBRs. Our experimental results confirm that this approximation hurts estimation accuracy. In addition, the accuracy of histogram-based methods is severely dependent on the bucketing resolution and the data distribution.

Spatial joins and related query optimization have been widely studied (Bouros & Mamoulis, 2019; Belussi et al., 2004; Vu et al., 2021; Xie et al., 2016; Taniguchi et al., 2022a, b). They mainly focus on how to execute complex spatial queries and select the most efficient join algorithms. Note that they have not answered how to estimate the cardinality of an intersection query.

RI (Georgiadis & Mamoulis, 2023), which focuses on efficient approximation of polygons for spatial joins, can be used for cardinality estimation. However, our evaluation in Fig. 2 finds it too slow to be used as a cardinality estimator. RI is designed for precise approximation of polygons and can provide an estimation near the true cardinality. However, it incurs nearly the same time cost of query execution, which is meaningless under the cardinality estimation context. (Note that cardinality estimation is conducted before query execution.)

Spatial intersection query has been widely studied for a long time (Bentley & Ottmann, 1979; Shamos & Hoey, 1976; Greiner & Hormann, 1998). The intersection of two polygons is determined by finding a pair of line segments intersecting with each other, where each segment belongs to a different polygon. Recent works (Frye & McKenney, 2022; Liu & Puri, 2020; Puri et al., 2013) mainly focus on accelerating intersection query processing by parallelization. These techniques are not available for our problem, because running an intersection query to estimate its result size is meaningless. Our problem is useful for making an efficient execution schedule of many intersection queries.

Learned cardinality estimation has shown remarkable improvements compared with traditional histogram-based methods (Wang et al., 2021b; Kipf et al., 2019; Hasan et al., 2020; Hu et al., 2022; Li et al., 2022). They can be categorized into data-driven methods (e.g., Yang et al., 2019; Meng et al., 2022) and query-driven methods(e.g., Han et al., 2021; Wang et al., 2021a).

Data-driven methods usually learn the data distribution using deep neural networks. Autoregressive models (Hasan et al., 2020; Yang et al., 2020) can approximate the joint distribution in a conditional manner and estimate the cardinality without any independent assumptions. However, data-driven methods cannot easily approximate the distribution of polygons because polygons have variable lengths and complex structures.

Query-driven methods are built on training queries and map training queries to the corresponding true cardinalities. The query-driven style is suitable for our task because it can ignore the complex data distribution of polygons, so PolyCard also takes this approach. Existing methods accept common data types in relational database systems, like integers, floats, and strings. These data have fixed sizes or can be easily encoded to a fixed size, whereas polygons do not have this observation. As mentioned in Section 2, the number of polygon vertices is variable and can be large. It is not trivial to transform polygons to a fixed size efficiently. Thus, existing learned cardinality estimators cannot deal with the polygon case.

3 PolyCard

3.1 Main idea

PolyCard is a learned cardinality estimator designed for intersection queries on polygons. We first solve the challenge of designing a learned model for polygons with variable-sized vertices. To map variable-sized vertices to fixed-sized ones, zero padding is the most direct solution and can make all polygons the same size as the maximal one. However, it increases the computation cost of NN, resulting in inefficiency. To transform polygons to a fixed size without having this inefficiency issue, we develop an adaptive sampling method. The idea behind the adaptive sampling is twofold. First, sampling can adjust the size freely. Second, adaptive sampling, which considers the spatial distribution of vertices, can well preserve the shapes of polygons even with smaller sizes of vertices.

Standard deep neural networks, such as convolutional neural networks (CNN), multi-layer perceptrons (MLPs), and autoregressive models, have been used for the cardinality estimation problem on the other data formats (Yang et al., 2019; Kipf et al., 2019; Negi et al., 2023; Sun et al., 2021). Inspired by these existing works, we build our neural network (NN) based on a combination of MLPs. Its structure helps achieve accurate estimation *within several microseconds* according to our evaluation, see Fig. 2.

Recall our discussion in Section 1.1: we want to train a NN with query polygons of evenly distributed cardinalities. (Training queries with more evenly distributed cardinalities can help reduce high-percentile errors.) Generating polygons from scratch is challenging, complex, and laborious, due to the variable number of vertices (Zhu et al., 1996) and complex and various polygon shapes. To overcome this challenge, we come up with the idea of using real-world datasets as sources of polygons. Specifically, we generate new (synthetic) query polygons based on real polygons from these datasets, with modifications such as shifting. Additionally, the cardinality distribution of the overall generated training data can be skewed and interfere with the convergence of model training, which makes the training a challenging task. We tackle this challenge by using a down-sampling strategy and achieve a uniform cardinality distribution for well model training. We develop a training data generator based on the above ideas.

3.2 Overview

Figure 3 shows the framework of PolyCard. For model training, we use our training data generator to obtain sufficient training queries and their true cardinalities. We featurize the training queries into dataset vectors and polygon vectors. Specifically, we represent the dataset information by one-hot vectors. Polygons are also represented as fixed-sized normalized vectors. A NN is trained with these vectors and the true cardinalities. After the training, we use the NN for cardinality estimation. Given a query, we featurize it into two vectors in the same way as the training. These two vectors are entered into the NN, and its output is $\widehat{card}$.

3.3 Training data generator

Our training data generator aims at two goals: (i) generating sufficient polygon queries and (ii) obtaining training data with evenly distributed cardinalities. Because polygons have variable sizes and vertices distribution, generating a polygon from scratch is not a good

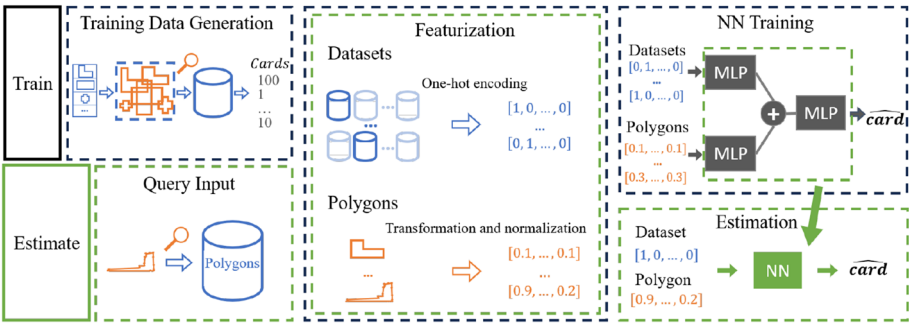


Fig. 3 Framework of PolyCard. **Training**: Our training data generator is used to obtain sufficient training queries and their true cardinalities (left-top part). We featurize the training queries into dataset vectors and polygon vectors (middle part), that is, we represent the dataset information by one-hot vectors, whereas polygons are repressed as fixed-sized normalized vectors. The NN is trained with these vectors and the true cardinalities (right-top part). **Cardinality estimation**: Given a query input, we featurize it into two vectors in the same way as the training (middle part). We then use the trained NN to obtain $\widehat{card}$

idea. We choose polygons from real-world datasets. Our generator chooses polygons from datasets randomly and performs transformations, i.e., shifting, reflection, and perturbation, to generate polygons serving as training queries, as shown in Fig. 4. To obtain a shifted version of a polygon P, we apply the same modifications to all coordinates of its vertices in each dimension. For the reflection, we randomly select a vertex of P and reflect all the other vertices vertically. To obtain a perturbation edition of a polygon P, we add a random small value to each coordinate of each vertex of P.

The distribution of polygons is naturally skewed (Macke et al., 2018). This observation makes the cardinality distribution of randomly generated queries have a long tail, as shown in Fig. 5(a). As discussed in Section 1.1, this distribution makes convergence of the training hard. Our generator helps obtain training data with evenly distributed cardinalities in two steps. First, generate new query polygons and compute their true cardinalities. Second, our generator guarantees the training data with an even cardinality distribution by down-sampling from the overall generated training data. To achieve this, we generate the cardinality histogram of training data. If the distribution shown in the histogram is skewed, the generator selects small portions of training data from buckets containing more data than the average. With our

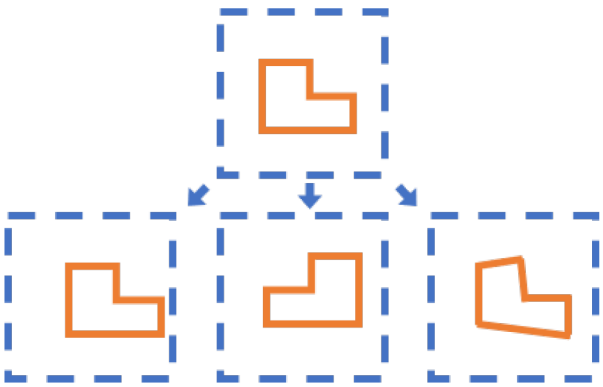


Fig. 4 Generation of polygons. We perform shifting, reflection, and perturbation on polygons from real-world datasets

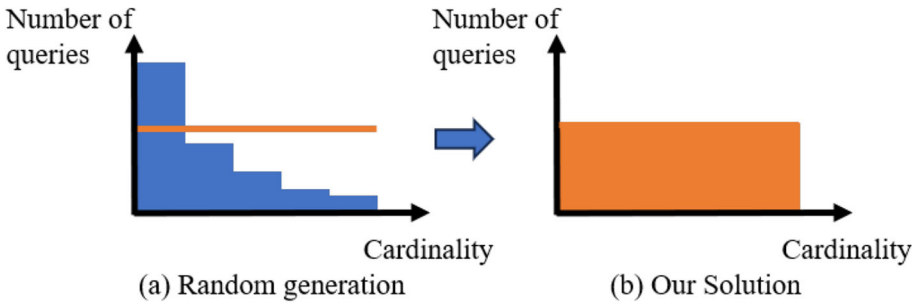


Fig. 5 Histogram of cardinalities of training queries on Sports dataset

Algorithm 1 POLYGON- TRANSFORMATION

Input: P (a polygon represented by an array of m vertices) and n (the number of vertices of the transformed polygon)
Output: P' (the transformed polygon)

$P' \leftarrow \emptyset, interval \leftarrow m/n, count \leftarrow 0$
if $n < m$ **then**
  $zone\_flag \leftarrow False$
  foreach $i \in [1, n]$ **do**
    $cur\_zone \leftarrow$ GET- ZONE($P[i]$) // $P[i]$ is the i-th vertex in P
    $count \leftarrow count + 1$
    if $zone\_flag[cur\_zone] = False$ **then**
      $P' \leftarrow P' \cup \{P[i]\}$
      $count \leftarrow count - interval$
      $zone\_flag[cur\_zone] \leftarrow True$
    else
      if $count \geq interval$ **then**
        $P' \leftarrow P' \cup \{P[i]\}$
        $count \leftarrow count - interval$
else
  foreach $i \in [1, m]$ **do**
    $count \leftarrow count + 1$
    $num\_vertices \leftarrow$ FLOOR($count/interval$)
    if $num\_vertices = 1$ **then**
      $P' \leftarrow P' \cup \{P[i]\}$
      $count \leftarrow count - interval$
    else
      if $num\_vertices \geq 2$ **then**
        $P' \leftarrow P' \cup \{$INTERPOLATION($P[i-1], P[i], num\_vertices$)$\}$
        $count \leftarrow count - interval \times num\_vertices$

generator, we can always obtain training data with the cardinality distribution like the one shown in Fig. 5(b).

3.4 Polygon transformation

The polygon transformation aims at transforming polygons with variable-sized vertices to a fixed size m, which makes polygons compatible for feeding into our NN model. Algorithm 1

shows the details of the transformation. Let m be the number of vertices of a polygon P. We want to transform P to a similar polygon with n vertices.

If $n < m$, we run an adaptive sampling that considers the distribution of vertices. We first compute the MBR of the original polygon and divide the MBR into four equal-sized zones according to the midpoints of the four edges of the MBR (line 5). Then, we perform a uniform-like sampling by scanning the vertices sequentially. During the scan, we guarantee that at least one vertex is sampled in each zone intersecting with the polygon. To achieve this, we locate the zone to which each vertex $P[i]$ belongs. If no sample belongs to the zone, we sample $P[i]$ (line 7).

If $m \geq n$, we adopt a uniform interpolation method for the transformation[2]. We sequentially scan the vertices and calculate the number of new vertices to be inserted between two vertices by interpolation (line 18). If the number of new vertices equals one, we simply treat the current vertex as a new vertex (lines 19–21). If the number of new vertices is larger than one, we interpolate the coordinates of two consecutive vertices and obtain the required number of new vertices with interpolated coordinates (lines 23–25). The transformation costs $O(\max(m, n))$ time.

In Fig. 6a, suppose that we transform the polygon to a fixed size of five vertices. Only two vertices fall into the left-bottom zone, and they are far away from the other vertices. If we run a standard random sampling, they are probably not sampled. Our adaptive sampling guarantees every zone where the vertices exit is covered. Thus, the transformation can preserve the shapes of polygons well. As shown in Fig. 6b, the shape of the transformed polygon is very similar to that of the original polygon.

3.5 Featurization

As shown in Fig. 3, the input data fed into the NN consists of two kinds of vectors, respectively representing the dataset information and the polygon. We featurize an input query into two vectors, a dataset vector and a polygon vector, to better represent the input and help the training converge.

The dataset vector represents which dataset is being queried. We use a one-hot vector of the length M, where M is the total number of datasets used as sources. For this vector, when the i-th dataset is accessed by a query, its corresponding dimension is one and the other dimensions are zero.

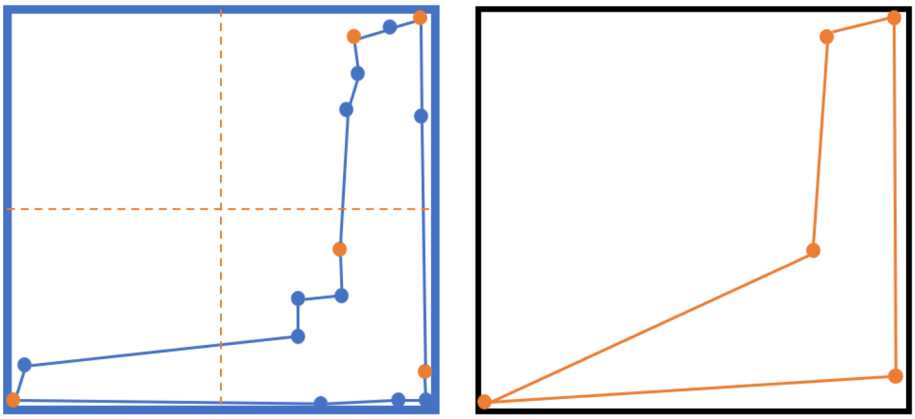
To featurize the query polygon, we first perform the transformation discussed in Section 3.4. Then we normalize the coordinates of vertices to avoid numerical instability. We construct the polygon vector by concatenating all coordinates of the polygon.

3.6 Model

For our NN, we employ MLPs, because they can approximate functions well and yield fast inferences (Collobert & Bengio, 2004; Mo et al., 2023). Each MLP is composed of two fully connected layers, each accompanied by a ReLU activation function:

$$MLP(x) = \max(0, \max(0, xW_1 + b_1)W_2 + b_2), \tag{1}$$

[2] Considering the estimation efficiency, n should be smaller than m for most polygons because a large n will increase the size of the NN and the computation cost for estimation.



(a) Original polygon (b) Transformed polygon

Fig. 6 Our polygon transformation. (a) The polygon is represented by vertices and edges connecting the vertices. The outside rectangle is the MBR of the polygon. The orange dashed line separates the MBR into four equal-sized zones referring to the four corners of the MBR. Given the fixed size of five vertices for transformation, we sample the orange vertices. (b) The transformed polygon

where x represents the input vector whereas W_1, W_2, b_1, and b_2 are learned parameters. The ReLU function is represented by $\max(0, x)$. We use one MLP for polygon vectors and another for dataset vectors. The outputs of the two MLPs are concatenated and then fed into the output MLP.

For the output MLP, we replace the final ReLU function with a sigmoid function, which is a common choice as the activation function for the output layer. Compared with the other NNs, the main advantage of our design is the computation speed. For example, the fully connected layer consumes much less computation time compared with the convolutional layer (widely used in various NNs).

Since PolyCard is a supervised estimator, we pre-compute the true cardinality of each training query polygon. Notice that the cardinality has a wide range of possible values. For training, we perform the logarithmization and normalization on the cardinality to compress the range and reduce the impact of extreme values on model training (Yu & Spiliopoulos, 2023). To get $\widehat{card}$ for a query, we perform the reversion of the logarithmization and the normalization on the output of the NN.

3.7 Training and estimation

Figure 3 shows the training and estimation flow of PolyCard. We take the following steps for the training: (i) generate training data by the generator, which contains the training queries and corresponding true cardinalities and (ii) train the NN with the training data. We use the q-error as our loss function, which is widely used for the cardinality estimation problem (Sun et al., 2021; Wu & Cong, 2021):

$$\text{q-error} = \frac{\max\left(\widehat{card}, card\right)}{\min\left(\widehat{card}, card\right)}.$$

To estimate the cardinality of a given query, we featurize this query into a dataset vector and a polygon vector. After feeding the featurized vectors into the NN, we obtain $\widehat{card}$ as discussed in Section 3.6. The estimation time is dependent only on the NN structure. Therefore, the estimation of PolyCard is stably fast for datasets of different sizes and queries of different selectivities.

4 Experimental evaluation

This section reports our experiment results and investigates the following questions:

- Is PolyCard accurate, fast, and stable?
- How good is PolyCard compared with competitors?
- Are the challenges mentioned in Section 1.1 severe problems for the cardinality estimation? How much does PolyCard benefit from solving those challenges?
- How much does PolyCard benefit from solving those challenges?

4.1 Setup

4.1.1 Dataset

We used four real-world datasets widely used in spatial data processing works[3] (Georgiadis & Mamoulis, 2023; Eldawy & Mokbel, 2015). They contain polygons representing the boundaries of various types of objects. Table 2 shows the information of the datasets.

4.1.2 Queries

We prepared one thousand randomly generated queries based on real-world polygons. We guaranteed that they have a wide cardinality distribution from one to tens of thousands.

4.1.3 Evaluated methods

We evaluated the following methods in this article.

- PolyCard: is our proposed method[4] The size of transformed polygons is set to ten by default. (The impact of different sizes is discussed in Section 4.3.) PolyCard is trained on a GPU.
- PostgreSQL[5]: refers to the cardinality estimation method used in PostgreSQL 16. It estimates the cardinality based on the statistics of the datasets including histograms. PostgreSQL does not provide the exact estimation time, so we used the planning time to represent its estimation time instead.
- CDHistogram (Jin et al., 2000): is one of the most representative histogram-based methods designed for spatial objects. It originally supports only rectangles. We extended it to

[3] https://spatialhadoop.cs.umn.edu/datasets.html

[4] Source code is available at: https://github.com/ji-yuchen/PolyCard..

[5] https://www.postgresql.org/docs/current/row-estimation-examples.html

Table 2 Statistics of datasets

Name	Cemetery	Sports	Parks	Buildings
Size (million)	0.19	1.8	10	115
Average #vertices per polygon	10.3	11.3	35.5	7.0

support polygons by using MBRs. The bucket size of CDHistogram on each dimension was set to 10,000 by default.

- RI (Georgiadis & Mamoulis, 2023): is a state-of-the-art approximation method for intersection joins on polygons. We employed its upper-bounding technique to estimate the cardinality of a given query's result.

4.1.4 Evaluated criteria

We used the q-error to measure the accuracy of each method, see Section 3.7. For estimation time, we report the average time to compute $\widehat{card}(R)$.

4.1.5 Environment

All experiments were conducted on a server with an Intel Xeon Gold 6254 CPU, a NVIDIA Quadro RTX 8000 GPU, and 768GB RAM. The OS was Ubuntu 22.04.4 LTS. PolyCard was implemented in Python with PyTorch. We implemented CDHistogram in C++. For RI, we used the C++ source code provided by the authors.

4.2 Overall performance

4.2.1 Estimation time

Figure 7 shows the estimation time of each method. PolyCard is the fastest method on all datasets except the Cemetery case. Furthermore, PolyCard provides a stable estimation time, i.e., its estimation time is not dependent on dataset size, which is desirable for large

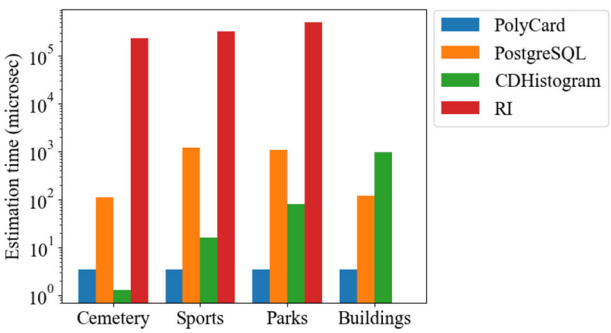


Fig. 7 Estimation time

Table 3 Accuracy evaluation results

Datasets	Methods	25th q-error	50th q-error	75th q-error	95th q-error	99th q-error
Cemetery	PolyCard	1.2	**1.45**	**1.91**	**3.22**	**5.3**
	PostgreSQL	**1.0**	1.85	2.33	5.0	7.0
	CDHistogram	1.6	2.25	3.26	6.0	8.0
Sports	PolyCard	**1.32**	**1.85**	**3.26**	**9.34**	**26.78**
	PostgreSQL	1.71	3.15	7.0	23.0	47.08
	CDHistogram	1.54	2.69	5.0	15.5	38.0
Parks	PolyCard	**1.29**	**1.79**	**3.03**	**8.86**	**19.73**
	PostgreSQL	1.46	2.35	4.5	18.69	90.26
	CDHistogram	2.47	3.43	5.31	13.0	30.02
Buildings	PolyCard	**1.72**	**2.99**	7.19	**37.47**	**126.36**
	PostgreSQL	7.74	28.85	75.68	333.05	1192.32
	CDHistogram	2.11	3.48	**6.48**	52.84	367.18

Bold shows the winner

datasets. On the other hand, CDHistogram becomes slower as the dataset size grows, and PostgreSQL is usually slower than CDHistogram. The estimation time of RI is much longer than the other methods and is longer than even the execution time of queries (normally several milliseconds)[6]. It is trivial that RI is not suitable for the estimation task. We therefore did not evaluate RI in the remaining experiments.

4.2.2 Accuracy

Table 3 shows the accuracy evaluation results. Overall, PolyCard achieves smaller errors than PostgreSQL and CDHistogram. When considering high percentile errors (i.e., 95th and 99th q-errors), which are of great concern for the cardinality estimation task, PolyCard beats the competitors on all datasets. PostgreSQL has the largest high percentile errors on all datasets except the Cemetery case. These large errors lead to a high risk of generating inefficient query plans.

4.3 Detailed analysis

4.3.1 How good is our polygon transformation?

We compare the sizes of input data obtained by the proposed polygon transformation solution (Section 3.4) with zero padding to confirm that zero padding is not a feasible approach. (Recall that zero padding is to fix the size of all polygons to the maximum one by adding zero.) Figure 8 observes that zero padding incurs too large data (more than 150GB) for NN

[6] The construction of RI on Buildings took more than several hours, so we omit its result.

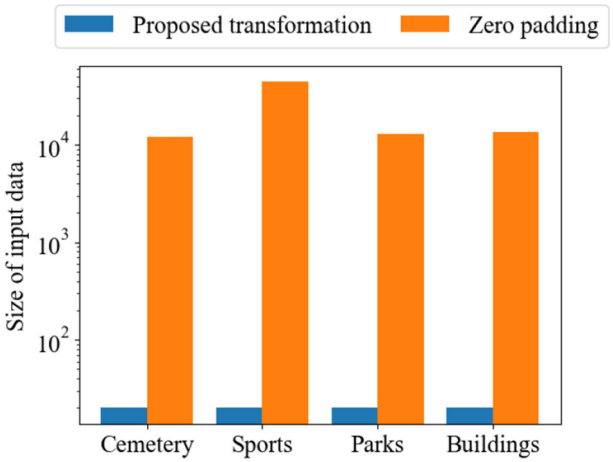


Fig. 8 Sizes of the input data for the NN achieved by the proposed polygon transformation and zero padding

training (i.e., tens of thousands of dimensions for each polygon). It is much larger than GPU RAM, and zero padding makes the training NN infeasible.

Furthermore, we compare our adaptive sampling transformation with uniform sampling and MBR approximation. Trivially, if method A provides a transformed polygon with a better similarity than method B, method A yields less error. Figure 9 shows the advantage of adaptive sampling. This result demonstrates that our polygon transformation is of high quality and helps achieve better accuracy.

4.3.2 Impact of reduced polygon size

Figure 10 shows the accuracy results corresponding to different sizes of transformed polygons. The accuracy can be greatly improved (specifically high percentile errors) when the size increases from 4 to 10. Considering the average number of vertices per polygon shown in 2, large sizes like 25 cannot further improve the performance. On the other side, large sizes increase the storage and computation costs. A grid-based search can help find the most suitable size for each dataset. Here, the choice of size = 10 is sufficient for our evaluation.

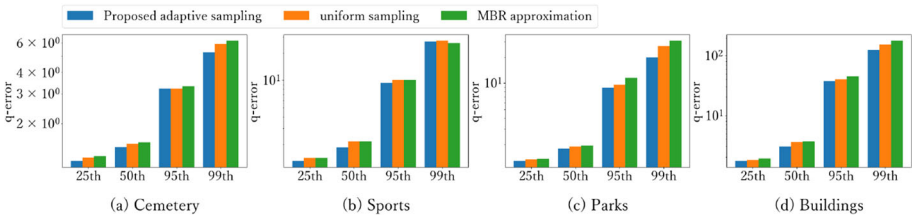


Fig. 9 Accuracy comparison of the proposed adaptive sampling, uniform sampling, and MBR approximation for polygon transformation

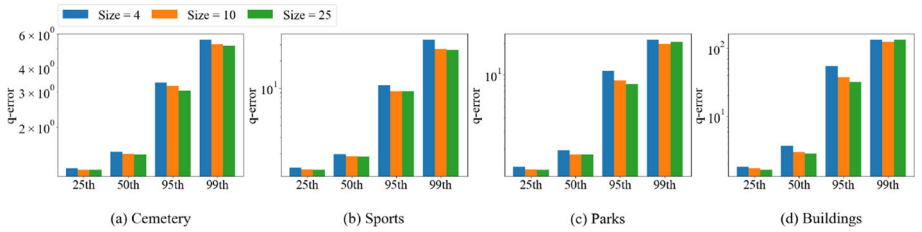


Fig. 10 Impact of sizes of transformed polygons on the accuracy

4.3.3 How good is our training data generator?

Figure 11 shows two convergence curves of PolyCard; one is generated by our proposed generator and the other is the random one discussed in Section 3.3. We can observe that PolyCard meets difficulty in convergence (at a small loss) when trained on randomly generated datasets. As discussed in Section 3.3, the cardinality distribution of randomly generated queries has a long tail. It makes convergence of the training hard. In the case of Cemetery, we found that more than sixty percent of queries in the randomly generated training queries have zero cardinality. As a result, the training loss shown in Fig. 11(a) fails to decrease and becomes a stubborn constant. Furthermore, as shown in Fig. 12(a), the trained model achieves poor estimation accuracy. Our training data generator solves this problem by generating training data with more evenly distributed cardinalities.

Figure 12 shows the estimation errors of PolyCard after the above training. The q-errors of PolyCard trained on randomly generated queries are much larger. The above results clearly show that our training data generator helps the training converge and achieve reasonable accuracy.

5 Conclusion

We presented PolyCard, which addressed the cardinality estimation problem of intersection queries on polygons. The main idea is twofold: (i) making learning models compatible with variable-sized polygons by applying adaptive sampling transformation on polygons and (ii) acquiring sufficient training data with evenly distributed cardinalities by a training data generator. Our experimental results show that our PolyCard has the following advantages compared with other competitors: (i) PolyCard improves 30% accuracy in comparison to other methods, (ii) PolyCard costs only 4 microseconds for a single estimation, and (iii) PolyCard is effective on datasets of different sizes and has low high percentile q-errors.

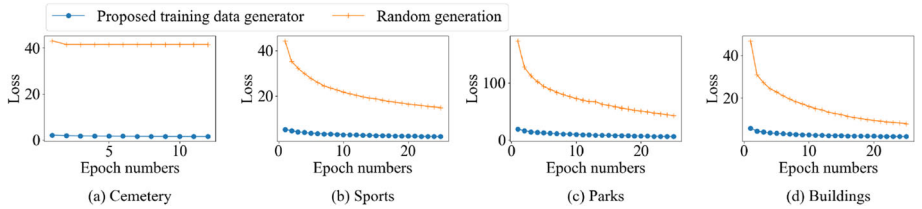


Fig. 11 Convergence of training

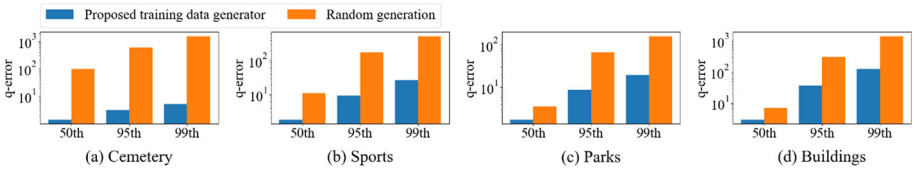


Fig. 12 The proposed training data generator helps achieve good accuracy: the estimation errors of PolyCard trained on the datasets generated by the generator and the random generation

Acknowledgements This work was partially supported by AIP Acceleration Research JPMJCR23U2, Japan Science and Technology Agency, and the Japan Society for the Promotion of Science KAKENHI Grant Number 23H03406 and 24K14961.

Author Contributions Y.J. and D.A. contributed to the conceptualization and methodology. Y.J. contributed to the software, evaluation, and the original draft. D.A. and Y.S. contributed to the manuscript writing. T.H. supervised the research.

Funding Open Access funding provided by Osaka University.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

Aji, A., Wang, F., & Saltz, J. H. (2012). Towards building a high performance spatial query system for large scale medical imaging data. In: SIGSPATIAL, pp 309–318, https://doi.org/10.1145/2424321.2424361

Amagata, D., & Hara, T. (2016). Monitoring maxrs in spatial data streams. In: EDBT, pp 317–328, https://doi.org/10.5441/002/edbt.2016.30

Amagata, D., & Hara, T. (2017). A general framework for maxrs and maxcrs monitoring in spatial data streams. *ACM Transactions on Spatial Algorithms and Systems, 3*(1), 1–34. https://doi.org/10.1145/3080554

Amagata, D., & Hara, T. (2019). Identifying the most interactive object in spatial databases. In: ICDE, pp 1286–1297, https://doi.org/10.1109/ICDE.2019.00117

Amagata, D., Arai, Y., Fujita, S., et al. (2022). Learned k-nn distance estimation. In: SIGSPATIAL, pp 1–4, https://doi.org/10.1145/3557915.3560935

Belussi, A., Bertino, E., & Nucita, A. (2004). Grid based methods for estimating spatial join selectivity. In: SIGSPATIAL, pp 92–100, https://doi.org/10.1145/1032222.1032238

Bentley, Ottmann. (1979). Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers, 100*(9), 643–647. https://doi.org/10.1109/TC.1979.1675432

Bouros, P., & Mamoulis, N. (2019). Spatial joins: what's next? *SIGSPATIAL Special, 11*(1), 13–21. https://doi.org/10.1145/3355491.3355494

Collobert, R., & Bengio, S. (2004). Links between perceptrons, mlps and svms. In: ICML, p 23, https://doi.org/10.1145/1015330.1015415

Derthick, M., Harrison, J., Moore, A., et al. (1999). Efficient multi-object dynamic query histograms. In: IEEE Symposium on Information Visualization, pp 84–91, https://doi.org/10.1109/INFVIS.1999.801862
Dutt, A., Wang, C., Nazi, A., et al. (2019). Selectivity estimation for range predicates using lightweight models. PVLDB 12(9):1044–1057. https://doi.org/10.14778/3329772.3329780
Eldawy, A., & Mokbel, M. F. (2015). Spatialhadoop: A mapreduce framework for spatial data. In: ICDE, pp 1352–1363, https://doi.org/10.1109/ICDE.2015.7113382
Frye, R., & McKenney, M. (2022). Per segment plane sweep line segment intersection on the gpu. In: SIGSPATIAL, pp 1–4, https://doi.org/10.1145/3557915.3561003
Georgiadis, T., & Mamoulis, N. (2023). Raster intervals: an approximation technique for polygon intersection joins. *Proceedings of the ACM on Management of Data, 1*(1), 1–18. https://doi.org/10.1145/3588716
Greiner, G., & Hormann, K. (1998). Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics, 17*(2), 71–83. https://doi.org/10.1145/274363.274364
Han, Y., Wu, Z., Wu, P., et al. (2021). Cardinality estimation in dbms: A comprehensive benchmark evaluation. PVLDB 15(4):752–765. https://doi.org/10.14778/3503585.3503586
Hasan, S., Thirumuruganathan, S., Augustine, J., et al. (2020). Deep learning models for selectivity estimation of multi-attribute queries. In: SIGMOD, p 1035–1050, https://doi.org/10.1145/3318464.3389741
Hashemi, M. (2019). Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. Journal of Big Data 6(1):1–13. https://doi.org/10.1186/s40537-019-0263-7
Hayashida, S., Amagata, D., Hara, T., et al. (2018). Dummy generation based on user-movement estimation for location privacy protection. *IEEE Access, 6*, 22958–22969. https://doi.org/10.1109/ACCESS.2018.2829898
Heckbert, P. S. (2013). *Graphics gems*. Elsevier.
Hori, K., Sasaki, Y., Amagata, D., et al. (2023). Learned spatial data partitioning. In: International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, pp 1–8, https://doi.org/10.1145/3593078.3593932
Hu, X., Liu, Y., Xiu, H., et al. (2022). Selectivity functions of range queries are learnable. In: SIGMOD, pp 959–972, https://doi.org/10.1145/3514221.3517896
Jacox, E. H., & Samet, H. (2007). Spatial join techniques. ACM Transactions on Database Systems 32(1):7–es. https://doi.org/10.1145/1206049.1206056
Ji, Y., Amagata, D., Sasaki, Y., et al. (2022). A performance study of one-dimensional learned cardinality estimation. In: DOLAP, pp 86–90
Ji, Y., Amagata, D., Sasaki, Y., et al. (2024). Safe: Sampling-assisted fast learned cardinality estimation for dynamic spatial data. In: DEXA, pp 201–216, https://doi.org/10.1007/978-3-031-68312-1_16
Jin, J., An, N., & Sivasubramaniam, A. (2000). Analyzing range queries on spatial data. In: ICDE, pp 525–534, https://doi.org/10.1109/ICDE.2000.839451
Kipf, A., Kipf, T., Radke, B., et al. (2019). Learned cardinalities: Estimating correlated joins with deep learning. In: CIDR
Kwon, S., Jung, W., & Shim, K. (2022). Cardinality estimation of approximate substring queries using deep learning. PVLDB 15(11):3145–3157. https://doi.org/10.14778/3551793.3551859
Leis, V., Gubichev, A., Mirchev, A., et al. (2015). How good are query optimizers, really? PVLDB 9(3):204–215. https://doi.org/10.14778/2850583.2850594
Li, B., Lu, Y., & Kandula, S. (2022). Warper: Efficiently adapting learned cardinality estimators to data and workload drifts. In: SIGMOD, pp 1920–1933, https://doi.org/10.1145/3514221.3526179
Liu, Y., & Puri, S. (2020). Efficient filters for geometric intersection computations using gpu. In: SIGSPATIAL, pp 487–496, https://doi.org/10.1145/3397536.3422264
Macke, S., Beutel, A., Kraska, T., et al. (2018). Lifting the curse of multidimensional data with learned existence indexes. In: Workshop on ML for Systems at NeurIPS, p 6
Mamoulis, N., Papadias, D., & Arkoumanis, D. (2004). Complex spatial query processing. *GeoInformatica, 8*, 311–346. https://doi.org/10.1023/B:GEIN.0000040830.73424.f0
Meng, Z., Wu, P., Cong, G., et al. (2022). Unsupervised selectivity estimation by integrating gaussian mixture models and an autoregressive model. In: EDBT, pp 247–259, https://doi.org/10.48786/edbt.2022.13
Mo, S., Chen, Y., Wang, H., et al. (2023). Lemo: A cache-enhanced learned optimizer for concurrent queries. *Proceedings of the ACM on Management of Data, 1*(4), 1–26. https://doi.org/10.1145/3626734
Negi P, Wu Z, Kipf A, et al (2023) Robust query driven cardinality estimation under changing workloads. PVLDB 16(6):1520–1533. https://doi.org/10.14778/3583140.3583164
Nishio, S., Amagata, D., & Hara, T. (2022). Lamps: Location-aware moving top-k pub/sub. *IEEE Transactions on Knowledge and Data Engineering, 34*(1), 352–364. https://doi.org/10.1109/ICDE55515.2023.00331
Pandey, V., van Renen, A., Kipf, A., et al. (2021). How good are modern spatial libraries? *Data Science and Engineering, 6*(2), 192–208. https://doi.org/10.1007/s41019-020-00147-9

Puri, S., Agarwal, D., He, X., et al. (2013). Mapreduce algorithms for gis polygonal overlay processing. In: IEEE International Symposium on Parallel & Distributed Processing, pp 1009–1016, https://doi.org/10.1109/IPDPSW.2013.254

Rigaux, P., Scholl, M., & Voisard, A. (2001). *Spatial databases: with application to GIS*. Elsevier.

Sasaki, Y. (2021). A survey on iot big data analytic systems: Current and future. *IEEE Internet of Things Journal, 9*(2), 1024–1036. https://doi.org/10.1109/JIOT.2021.3131724

Shamos, M. I., & Hoey, D. (1976). Geometric intersection problems. In: Annual Symposium on Foundations of Computer Science, pp 208–215, https://doi.org/10.1109/SFCS.1976.16

Shekelyan, M., Dignös, A., Gamper, J., et al. (2021). Approximating multidimensional range counts with maximum error guarantees. In: ICDE, pp 1595–1606, https://doi.org/10.1109/ICDE51399.2021.00141

Shi, Y., Tong, Y., Zeng, Y., et al. (2023). Efficient approximate range aggregation over large-scale spatial data federation. *IEEE Transactions on Knowledge and Data Engineering, 35*(1), 418–430. https://doi.org/10.1109/TKDE.2021.3084141

Sun, J., Li, G., & Tang, N. (2021). Learned cardinality estimation for similarity queries. In: SIGMOD, pp 1745–1757, https://doi.org/10.1145/3448016.3452790

Taniguchi, R., Amagata, D., & Hara, T. (2022a). Efficient retrieval of top-k weighted spatial triangles. In: DASFAA, pp 224–231, https://doi.org/10.1007/978-3-031-00123-9_17

Taniguchi, R., Amagata, D., & Hara, T. (2022). Efficient retrieval of top-k weighted triangles on static and dynamic spatial data. *IEEE Access, 10*, 55298–55307. https://doi.org/10.1109/ACCESS.2022.3177620

Vu, T., Belussi, A., Migliorini, S., et al. (2021). A learned query optimizer for spatial join. In: SIGSPATIAL, pp 458–467, https://doi.org/10.1145/3474717.3484217

Wang J, Chai C, Liu J, et al (2021a) Face: A normalizing flow based cardinality estimator. PVLDB 15(1):72–84. https://doi.org/10.14778/3485450.3485458

Wang, X., Qu, C., Wu, W., et al. (2021b). Are we ready for learned cardinality estimation? PVLDB 14(9):1640–1654. https://doi.org/10.14778/3461535.3461552

Wang, Y., Xiao, C., Qin, J., et al. (2020). Monotonic cardinality estimation of similarity selection: A deep learning approach. In: SIGMOD, pp 1197–1212, https://doi.org/10.1145/3318464.3380570

Woltmann, L., Olwig, D., Hartmann, C., et al. (2021). Postcenn: postgresql with machine learning models for cardinality estimation. PVLDB pp 2715–2718. https://doi.org/10.14778/3476311.3476327

Wu, P., & Cong, G. (2021). A unified deep model of learning from both data and queries for cardinality estimation. In: SIGMOD, pp 2009–2022, https://doi.org/10.1145/3448016.3452830

Xie, D., Li, F., Yao, B., et al. (2016). Simba: Efficient in-memory spatial analytics. In: SIGMOD, pp 1071–1085, https://doi.org/10.1145/2882903.2915237

Yang Z, Liang E, Kamsetty A, et al (2019) Deep unsupervised cardinality estimation. PVLDB 13(3):279–292. https://doi.org/10.14778/3368289.3368294

Yang, Z., Kamsetty, A., Luan, S., et al. (2020). Neurocard: One cardinality estimator for all tables. PVLDB 14(1):61–73. https://doi.org/10.14778/3421424.3421432

Yosinski, J., Clune, J., Nguyen, A., et al. (2015). Understanding neural networks through deep visualization. In: Deep Learning Workshop on International Conference on Machine Learning, pp 448–456

Yu, J., & Spiliopoulos, K. (2023). Normalization effects on deep neural networks. *Foundations of Data Science, 5*(3), 389–465. https://doi.org/10.3934/fods.2023004

Zhu, C., Sundaram, G., Snoeyink, J., et al. (1996). Generating random polygons with given vertices. *Computational Geometry, 6*(5), 277–290. https://doi.org/10.1016/0925-7721(95)00031-3