



Title	SAFE: Sampling-Assisted Fast Learned Cardinality Estimation for Dynamic Spatial Data
Author(s)	Ji, Yuchen; Amagata, Daichi; Sasaki, Yuya et al.
Citation	Lecture Notes in Computer Science. 2024, 14911, p. 201-216
Version Type	AM
URL	https://hdl.handle.net/11094/105171
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

SAFE: Sampling-assisted Fast Learned Cardinality Estimation for Dynamic Spatial Data

Yuchen Ji¹, Daichi Amagata¹, Yuya Sasaki¹, and Takahiro Hara¹

Osaka University, Osaka, Japan

{ji.yuchen, amagata.daichi, sasaki, hara}@ist.osaka-u.ac.jp

Abstract. Cardinality estimation for spatial queries plays an important role in query scheduling and optimization. Spatial datasets are fully dynamic, and this setting necessitates an update-friendly, low-latency, and accurate cardinality estimator. However, existing cardinality estimation methods suffer from time-consuming updates and/or inefficient estimation. This work proposes SAFE (Sampling-Assisted Fast learned cardinality Estimator), which is carefully designed for dynamic spatial data. We specifically develop a sampling strategy that uses a quad-tree-based data partitioning and extracts a small subset, to enable fast training of cardinality estimation models. In addition, we employ 2-tier regression models to approximate the spatial data distribution while achieving accurate and fast cardinality estimation. We furthermore provide an incremental model update strategy to avoid re-training all models from scratch when we receive updates. We conduct experiments on real and synthetic datasets. Their results demonstrate that SAFE (i) outperforms state-of-the-art cardinality estimation models and (ii) efficiently handles data updates while ensuring accurate and low-latency estimation.

Keywords: Database · Cardinality estimation · Machine learning.

1 Introduction

Nowadays, substantial amounts of spatial points are being generated, because of the widespread use of GPS-enabled mobile devices [4, 17]. Due to this trend, spatial data applications require data management technologies that can offer efficient and high-quality analysis and search for spatial data [2]. One of the fundamental technologies for spatial data is *range queries*, which find spatial points within a specified range.

Reducing the latency of dealing with workloads consisting of range queries is essential to provide comfortable applications for users. Cost-based query optimization is often used to obtain the most efficient query plan. Cardinality estimation, which estimates the result size of a given query, is one of the most important core functions for query optimization. Because spatial data applications usually need to deal with a significant number of range queries, the accuracy and efficiency of cardinality estimation for range queries are directly related to the efficiency of dealing with a workload of range queries [11].

Spatial datasets are generally dynamic, i.e., they allow frequent updates, including insertions and deletions [1, 14]. Therefore, cardinality estimators should be updatable according to dataset updates. This is a trivial requirement: if a given dataset is subject to updates, static estimators cannot consider the latest data distribution and the estimation results will be inaccurate. This paper addresses the cardinality estimation problem on dynamic spatial data and designs a fast, accurate, and efficiently-updatable cardinality estimator.

Motivation and challenge. Recent results of *learned* cardinality estimation methods show remarkable improvements compared with traditional methods (e.g., histogram-based methods) [7, 12]. Learned cardinality estimation methods can be categorized into *query-driven* models and *data-driven* models.

Query-driven models are built on training queries and map a training query to its true cardinality. Most of them use deep learning to capture the relationship between queries and cardinalities [15, 18]. The nature of “relying on training queries” is infeasible for dynamic data, because we need to collect training queries and compute their true cardinality whenever we receive updates.

Data-driven models usually learn the distribution of a given dataset mainly through deep neural networks [12, 21]. They hence take a long time to build a model (at least several “minutes” of training time are required for large datasets) [7]. This property limits to only static data, as they cannot support fast model updates (i.e., re-training is required whenever datasets are updated). LHist [11] is a state-of-the-art cardinality estimator for spatial data and does not use complex models like neural networks, but it also does not support model updates. Traditional methods, such as histogram and sampling, also can be considered as data-driven models, because they generate some statistics from original datasets. Although their structures are easy to build, their accuracy is lower than those of learning-based models [6, 15].

We here summarize the challenges that need to be solved to deal with dynamic spatial data effectively. (1) **Stable performance:** The estimation accuracy and speed should be stable even if a given dataset is updated. (2) **Fast estimation:** Cardinality estimation is frequently called, so fast estimation (e.g., microsec-level) is desirable. (3) **Update-friendly:** Estimation models should be easily (efficiently) updatable according to data updates. As seen above, existing approaches cannot satisfy these requirements at the same time.

Contribution. We solve the above challenges and make the following contributions to the cardinality estimation problem on dynamic spatial data.

(i) **SAFE.** We propose SAFE, a sampling-assisted fast and accurate learned cardinality estimator. Our estimator specializes in solving frequent data updates while keeping efficient and accurate cardinality estimations. It enables fast training by a sampling strategy using data partitioning based on a quad-tree. Our key observation is that we can accelerate the model updates from two perspectives: (i) updating only the necessary models and (ii) fast training. (As shown later, a SAFE consists of multiple light-weighted models.)

As for the first perspective, existing cardinality estimation methods typically need to update/train their whole structures/models from scratch whenever they

Table 1: Overview of symbols frequently used in this paper

Symbol	Description
P	Spatial dataset consisting of N points
R	Range query
$card(R)$ ($\widehat{card(R)}$)	True (estimated) cardinality of R
f_x	Regression model for the x-dimension
f_y^i	i -th regression model for the y-dimension

receive an update. To avoid this issue, we employ 2-tier regression models, based on our insight that each model can be updated independently. In addition, there are various regression models, and their performances depend on the data distributions. To make an appropriate selection among these models, we design a cost model that considers accuracy, estimation time, and construction time. This cost model helps select a model with a balanced trade-off between time and accuracy.

Next, as for fast training, we propose a sampling strategy using data partitioning based on a quad-tree. A smaller sample size alleviates the training time while learning the data distribution, as samples follow the data distribution. Thanks to our idea of these two perspectives, SAFE provides stable performance.

(ii) Incremental model update strategy for fast dealing with data updates. To enable fast model updates, we propose an incremental model update algorithm for SAFE. The SAFE structure can avoid model updates from scratch, and, for a given update (e.g., point insertion), only several regression models are affected. The SAFE structure enables to find such models quickly, and only affected models are updated.

(iii) Experiments. We conduct experiments on both real-world and synthetic datasets. Our results demonstrate that SAFE outperforms state-of-the-art cardinality estimation methods and can efficiently handle dataset updates.

2 Preliminary

Problem Statement. Let P be a dataset of N two-dimensional (i.e., geospatial) points, and each point p in P is represented as $p = (x, y)$. Table 1 summarizes the symbols used throughout this paper.

▷ *Query predicate.* This paper considers orthogonal range queries. Let R be the specified range of a given orthogonal range query, and the orthogonal range *reporting* query outputs all points in P that overlap R . A range R can be represented by its left-bottom point l and right-top point u . For conciseness, we use “range queries” to denote orthogonal range queries. Also, R is used to denote a range query hereinafter.

▷ *Cardinality* of a range query R is the number of points in P that overlap R . We use $card(R)$ to denote the cardinality of R . The notion of *selectivity* is represented as $\frac{card(R)}{N}$.

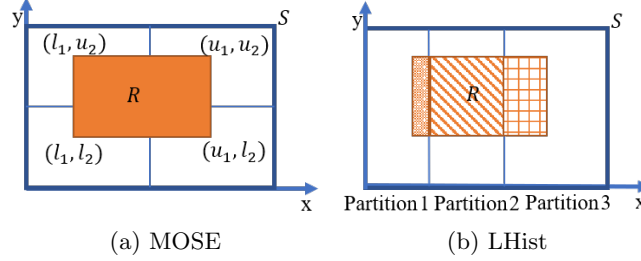


Fig. 1: Solutions for approximating data distributions. (a) MOSE approximates the distribution of multi-dimensional points by using the lattice regression model. (b) LHist recursively learns the data distribution of each dimension, similar to an equal-depth histogram.

We address the cardinality *estimation* problem, which is to infer $\text{card}(R)$. We use $\widehat{\text{card}}(R)$ to denote an estimated cardinality of R . Our objective is to compute $\widehat{\text{card}}(R)$ efficiently and accurately. Although there exists an $O(\sqrt{N})$ time range *counting* algorithm (i.e., *kd-tree* [3]), it requires many point accesses and is slow. We therefore consider a learning-based approach, because it achieves $O(1)$ time estimation (the estimation time is dependent only on a model). We assume that the query workload is unknown for generality.

Approximating Data Distribution for Cardinality Estimation.

To estimate the cardinality, data-driven methods approximate the joint multi-dimensional data distribution. As shown in Fig. 1a, MOSE [16] approximates the multi-dimensional CDF by the lattice regression model:

$$\widehat{\text{card}}(R) = f_{MOSE}(u_1, u_2) - f_{MOSE}(u_1, l_2) - f_{MOSE}(l_1, u_2) + f_{MOSE}(l_1, l_2),$$

where f_{MOSE} approximates the joint CDF of the two-dimensions. However, as a query-driven method, it relies heavily on training queries. This approach cannot be used for unknown query workloads and dynamic data.

Another way is to approximate the data distribution on *each* dimension. LHist [11] partitions the data space into multi-dimensional grid cells and employs hierarchical regression models, which is shown in Fig. 1b. The estimated cardinality is computed by:

$$\widehat{\text{card}}(R) = \sum_i f_{LHist}^i(D_i),$$

where $f_{LHist}^i(D_i)$ approximates the cardinality of the i -th partition D_i . This recursive way can adapt to frequent updates, because each regression model can be trained independently. By selecting only the models that need to be updated, we can avoid training all models from scratch and efficiently keep accurate models even on dynamic data. However, LHist does not have this idea and support model updates. SAFE implements this new idea and employs 2-tier regression models in an update-friendly manner.

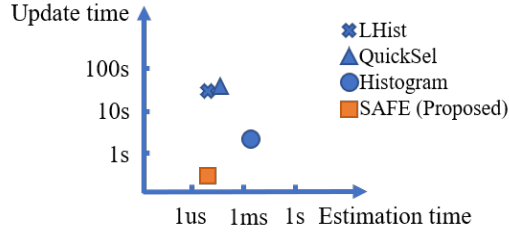


Fig. 2: Comparison of estimation and update times of cardinality estimation methods on a Gaussian dataset consisting of 20 million points

Update Solutions of Existing Methods. The most straightforward approach for model updates is to re-train all models. However, it is laborious, especially for frequent updates. Fig. 2 compares the update and estimation times on a Gaussian dataset with 20 million two-dimensional points. For LHist and Histogram, they need to reconstruct all models, so their update times are equal to the construction times. A state-of-the-art query-driven model, QuickSel [15], builds its models from scratch based on records of queries and true cardinalities. However, the update of QuickSel follows the same procedure with its construction, which is very slow. It can be seen that the existing learning-based methods (i.e., LHist and QuickSel) incur a long reconstruction time for updates, which is not acceptable in the dynamic setting.

We overcome this drawback and design our solution, SAFE, that supports incremental model updates and achieves fast estimations and updates. Fig. 2 depicts that SAFE satisfies fast estimation and update times.

3 SAFE

Main idea. SAFE is a learned cardinality estimator designed for dynamic spatial data. To accurately learn the 2-dimensional CDF, we employ 2-tier regression models, each of which is a simple (e.g., polynomial) model (see Section 2 for an intuition of using CDF). The first tier (resp. second tier) corresponds to the x (resp. y)-dimension¹. The idea behind employing simple regression models is twofold. First, they can be trained quickly, whereas complex regression models (e.g., deep neural networks) do not have this property. Second, by dividing the x -dimensional space into disjoint buckets, we can focus on the y -dimensional CDF on each bucket, and the CDF tends not to have a complex curve. More precisely, the first tier model is responsible for dividing the data space into disjoint subspaces based on x -dimensional values, whereas the second tier models are responsible for cardinality estimation.

The accuracy of learned cardinality estimator becomes worse on dynamic data if models are not updated. As we assume frequent updates (e.g., insertions and deletions), cardinality estimators should be robust against this dynamic

¹ The order of the x - and y -dimensions is changeable.

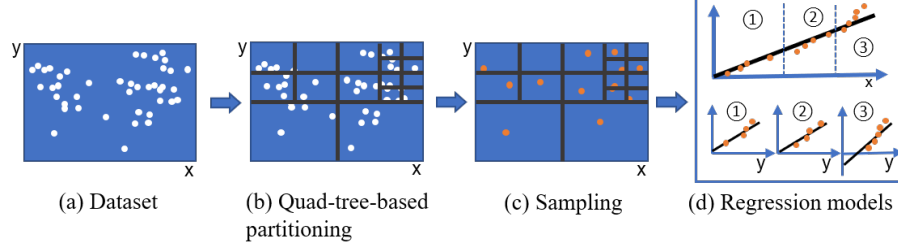


Fig. 3: Construction of a SAFE. (a) Given a dataset P , (b) we partition P into disjoint subspaces by using quad-tree-based partitioning. (c) We sample a random point from each partition, and (d) train 2-tier regression models by using the samples. The regression model on the x-dimension (the first tier) partitions P into m buckets based on the x-dimensional values, and a regression model is created for each bucket (the second tier).

setting. Then, we come up with two ideas: (i) fast training with small samples and (ii) updating only the necessary models. To implement these ideas, we propose to use “good” samples for accurately training the regression models, and we design the structure of SAFE to easily identify which model has to be updated. With these techniques, we can incrementally update only models that need to be updated even when we receive dataset updates.

Overview. Fig. 3 shows an overview of SAFE construction. We first obtain samples by using data partitioning based on a quad-tree. The samples are used to fast train regression models. We then train 2-tier regression models (the first tier corresponds to the CDF on the x-dimension while the second tier corresponds to the CDF on the y-dimension). Note that each cardinality estimation model for the corresponding subspace does not affect the others, thereby it can be trained independently. Fig. 3(d) illustrates how (i) the data space is divided into multiple (three in the figure) buckets based on the x-dimensional values and (ii) a model in each bucket is trained while considering the y-dimensional CDF. Algorithm 1 describes the pseudo-code of our construction algorithm.

3.1 Sampling by Data Partitioning based on Quad-tree

When a model needs to be updated, fast training is required for dynamic data. To this end, we consider using samples. That is, we train our regression models from samples obtained from a given dataset. This is a simple-yet-effective approach, as this approach makes the training easier and faster. We here note that straightforward random sampling is not suitable, as it may fail to sample points in sparse regions, leading to wrong learning.

To avoid this issue, we partition the whole data space into disjoint fine-grained subspaces so that we can sample a point from each subspace. We achieve this by using quad-tree-based partitioning. A quad-tree decomposes a given data space into equal-sized four disjoint subspaces [5]. This is repeated until a given space has a small number of points. (We set the cell capability of the quad-tree

Algorithm 1: CONSTRUCTION

Input: P (a spatial dataset) and m (the number of buckets)
Output: $f_x, f_y^1, \dots, f_y^m$ (regression models)

- 1 $P' = \{P_1, P_2, \dots\} \leftarrow \text{PARTITION-DATASET}(P)$ // based on a quad-tree
- 2 $S \leftarrow \text{GET-SAMPLES}(P')$ // sample a random point from each partition
- 3 $P_x \leftarrow$ a set of x-dimensional values in S
- 4 $f_x \leftarrow \text{TRAIN}(P_x)$ // learning the CDF curve in x-dimension by least squares
- 5 $B_1, B_2, \dots, B_m \leftarrow \emptyset$
- 6 **foreach** $p \in S$ **do**
- 7 $B_j \leftarrow B_j \cup \{p\}$, where $j = \lfloor \frac{m}{N} f_x(x_i) \rfloor$
- 8 **foreach** $i \in [1, m]$ **do**
- 9 $P_y^i \leftarrow$ a set of y-dimensional values in B_i
- 10 $f_y^i \leftarrow \text{TRAIN}(P_y^i)$ // learning the CDF curve in y-dimension by least squares

to 10 in our experiments.) As seen from Figs. 3(b) and (c), this approach does not fail to sample points from sparse regions. We pick one random sample from each partition. Note that we do not take samples from empty partitions.

3.2 Regression Models for Cardinality Estimation

Considering data-driven models for cardinality estimation without relying on queries, existing works employ hierarchical regression models. However, they do not provide any ideas of how we efficiently update such models or they cannot avoid re-training all models from scratch [11, 16]. Our structure is designed based on the idea that each regression model can be independently updatable. In addition, we design a cost model to select a suitable model for a given dataset to achieve a good trade-off between accuracy and estimation time.

Layout & Estimation. As shown in Fig. 3(d), we employ 2-tier regression models. For the x-dimension, we divide the data space into m subspaces called buckets, based on a regression model that approximates the x-dimensional CDF. On each bucket, we train a regression model to approximate the y-dimensional CDF curve of points falling in this bucket. The first tier has a single model f_x , while the second tier has m regression models $f_y^1, \dots, f_y^m$.

To estimate the cardinality of a query R , the first tier model is used to find the buckets overlapping R . Fig. 4 illustrates the estimation flow. Because the model for the x-dimension, f_x , divides the data space into m buckets, the left-most and right-most buckets overlapping R are respectively computed by $\lfloor \frac{m}{N} f_x(l_1) \rfloor$ and $\lfloor \frac{m}{N} f_x(u_1) \rfloor$, where l_1 and u_1 are respectively the left-most and right-most x-dimensional values of R . Then, we estimate the cardinality by considering a set B of all overlapping buckets:

$$\widehat{\text{card}}(R) = \alpha \sum_{i \in B} (f_y^i(u_2) - f_y^i(l_2)) \quad (1)$$

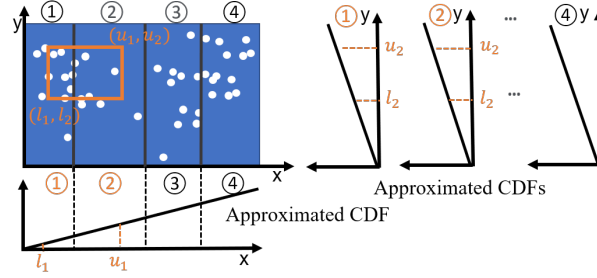


Fig. 4: Illustration of the two-dimensional layout for regression models. For the x-dimension, a regression model divides the data space into m buckets according to the data values of the x-dimension ($m = 4$ in the figure). Then, regression models are built on each bucket to approximate the y-dimensional CDF. For a query R , the regression model for the x-dimension locates the buckets that need to be accessed. The estimated $\widehat{card}(R)$ is obtained by summing the results from all the accessed buckets.

Algorithm 2: ESTIMATION

Input: $f_x, f_y^1, \dots, f_y^m$ (regression models) and R (a range query)
Output: $\widehat{card}$
1 $\widehat{card} \leftarrow 0, \mathcal{B} \leftarrow \text{GET-BUCKETS}(f_x, R)$ // get a set of all buckets overlapping R
2 **foreach** $B_i \in \mathcal{B}$ **do**
3 $\widehat{card} \leftarrow \widehat{card} + (f_y^i(u_2) - f_y^i(l_2))$
4 $\widehat{card} \leftarrow \alpha \times \widehat{card}$ // α is the average number of points in each partition

Table 2: Types of regression models (θ_i is a parameter and $z = x$ or $z = y$)

Name	Description
Polynomial	$f(z) = \theta_0 + \theta_1 z + \dots + \theta_n z^n$
Spline Linear	$f(z) = \begin{cases} \theta_0 + \theta_1 z & (z \leq \theta_3) \\ \theta_2 z + \theta_1 \theta_3 + \theta_0 & (z > \theta_3) \end{cases}$
Log Linear	$f(z) = \exp(\theta_0 z + \theta_1)$

where l_2 and u_2 are respectively R 's bottom and top values in the y-dimension, and α is the average number of points in each partition obtained by the quad-tree-based partitioning. Recall that our regression models are trained on *sample* points (i.e., a small subset of P). Therefore, the cardinality estimated by f_y needs to be scaled up, and α is used to achieve this.

For computing $\widehat{card}(R)$, we access at most $1 + m = O(1)$ regression models, and each model needs $O(1)$ time, see Table 2. Therefore, Equation (1) needs $O(1)$ time. A pseudo-code of our estimation algorithm appears in Algorithm 2.

Selection of Regression Models. Regression models are responsible for approximating one-dimensional CDFs. Considering the variety of real data distri-

bution, it is better to consider which model we should select. However, automatically selecting it is challenging.

To address this task, we propose a strategy to select the most suitable type of regression model based on the following three parts. (Table 2 lists the representative types of regression models, which are candidates for SAFE.)

- **Estimation error.** It is not practical to obtain estimation errors during construction. Instead, we measure the least square training error of a given regression model, because it represents how the model approximates the data distribution.
- **Estimation time.** Fast estimations highly rely on regression models with low computation time. Considering the types of candidates, it is easy to get an expected estimation time by feeding in default values during construction.
- **Training time.** The dynamic setting asks for frequent and efficient model updates, so the training time is also important. Measuring the training time of a given model is trivial.

Then, we measure the cost for a given regression model f as follows:

$$Cost(P, f) = E_{train} + T_{est} + T_{train}, \quad (2)$$

where E_{train} , T_{est} , and T_{train} represent the estimation error, estimation time, and training time, respectively. We collect E_{train} , T_{est} , and T_{train} of all candidates, and each of them is normalized to $[0, 1]$ (so $Cost(P, f) \in [0, 3]$)². Based on the costs, we use the model with the lowest cost. This cost model evaluation is done for each dimension.

Training of Regression Models. For a given regression model, we use the least square method to obtain its parameters. We introduce the training of polynomial models. (The training for spline linear and log linear models is essentially similar to the training for a linear regression model, which is trivial.)

Assume that we have a training dataset containing pairs of (p_i, r_i) , where r_i is the rank of p_i in the dataset determined by the sorted order of a given dimension (x or y). For a polynomial model $f(z) = \theta_0 + \theta_1 z + \dots + \theta_n z^n$, where z is the x- or y-dimensional value of a given point, the residual sum of squares is

$$RSS = \sum_{i=1}^{N'} (f(z_i) - r_i)^2, \quad (3)$$

where $N' \leq N$ is the number of input points. RSS can be written in a matrix format [10]:

$$RSS = (Z_v \theta - Q_\gamma)^T (Z_v \theta - Q_\gamma), \quad (4)$$

where

$$Z_v = \begin{bmatrix} 1 & z_1 & z_1^2 & \cdots & z_1^n \\ 1 & z_2 & z_2^2 & \cdots & z_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_{N'} & z_{N'}^2 & \cdots & z_{N'}^n \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}, Q_\gamma = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{N'} \end{bmatrix}. \quad (5)$$

² It is also possible to use weights $\{w_1, w_2, w_3\}$ for the criteria.

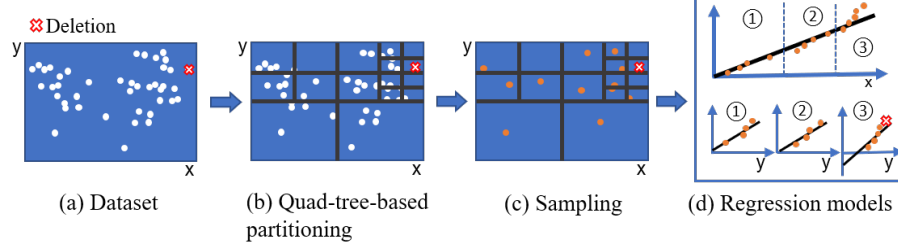


Fig. 5: The workflow of model update. The red token represents a point deletion.

Here Z_v is the Vandermonde matrix. The optimal $\theta = \{\theta_0, \theta_1, \dots, \theta_n\}$ should minimize RSS . Hence, we can calculate θ by:

$$\theta = (Z_v^T Z_v)^{-1} Z_v^T Q_\gamma. \quad (6)$$

Recall that our flow of training each model (training f_x then f_y^i) is described in Algorithm 1.

3.3 Model Update

We consider point insertions, deletions, and replacements as updates. Fig. 5 illustrates the flow of our model update when a deletion is given.

Given an update, we first check whether we need to update the quad-tree-based partitions, which is shown in Fig. 5(b). When necessary, the samples also need to be updated. (If not, we update α in Equation (1), and the update is over.) Specifically, we need to update the samples from the corresponding partitions in the following three cases:

- A partition is split (mainly due to an insertion).
- A partition is merged (mainly due to a deletion).
- The current sample is removed (due to a deletion or replacement).

In these cases, we create update records. Specifically, we create $\langle s, ins \rangle$ for an insertion and $\langle s, del \rangle$ for a deletion, where s is the original sample of the partition that needs to be updated. Also, each of new partitions picks a random sample. These new samples are assigned to the corresponding buckets. Notice that replacements are regarded as deletion-then-insertion.

As each sample s is associated with a unique bucket, we can identify the bucket to which s belongs by using a hash table, i.e., by using s as a key. From this approach and update records, we can identify the models that need to be updated, as each model is trained on the samples in the bucket. We update the corresponding regression models by using the training method (in Section 3.2).

We note that, given an update, a quad-tree partition rarely changes, so we rarely update the models. Even if some partitions are updated, we need to update only a small number of models, as such updated partitions exist locally. This mechanism helps minimize the time for updating the SAFE structure, which is empirically confirmed in Section 4.3.

4 Experiments

This section reports our experimental results. We conducted experiments on a server running Ubuntu 18.04 LTS and equipped with an Intel Xeon Gold 6254@3.10GHz CPU and 768GB RAM. All methods were single-thread and implemented in C++.

4.1 Setup

Datasets. We used three real-world datasets (Colorado, Paris, and Shikoku) and one synthetic dataset (Gaussian). Colorado, Paris, Shikoku, and Gaussian have 35, 27.5, 9, and 20 million points, respectively. The three real-world datasets contain pairs of latitudes and longitudes extracted from OpenStreetMap (OSM)³. Gaussian was generated by using a Gaussian distribution with a mean of 100 and a variance of 500 for each dimension.

Cardinality estimation methods. Our experiments compared SAFE with the following cardinality estimation methods.

- LHist [11]: is the state-of-the-art cardinality estimation method for spatial data. As recommended in [11], we set the degree of polynomial models to 6.
- QuickSel [15]: is a state-of-the-art supervised (query-driven) cardinality estimation method. It continuously learns from incoming queries to update its Gaussian mixture models. To make it comparable to the above methods, we prepared 1,000 queries as the training set for initializing QuickSel.

Evaluation criteria. To measure the accuracy of each method, we used q-error:

$$\text{q-error} = \frac{\max(\widehat{\text{card}}(R), \text{card}(R))}{\min(\widehat{\text{card}}(R), \text{card}(R))}. \quad (7)$$

It represents the factor between the estimated and the true cardinality, and it is the most widely used metric for the cardinality estimation problems [9, 20]. For the estimation time, we report the average time to compute $\widehat{\text{card}}(R)$.

4.2 Result: Static Data Case

For each dataset, we prepared three groups of queries. Each group contains 1,000 randomly generated range queries with a mix of selectivity of about 0.1 and 0.01. The three groups are regarded as disjoint clusters, i.e., queries in the same group (in different groups) have similar (totally different) query regions. In our preliminary experiment, we found that R-tree runs range counting with a selectivity of 0.01 in less than 10 milliseconds in average. Then, to ensure the effectiveness of estimations, we controlled the estimation time of all the evaluated methods to around 100 microseconds in the following experiments by default.

³ <http://download.geofabrik.de/>

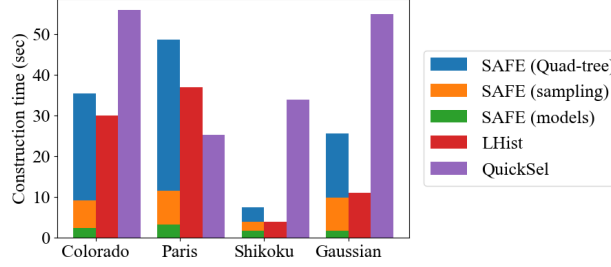


Fig. 6: Construction time evaluation on different datasets

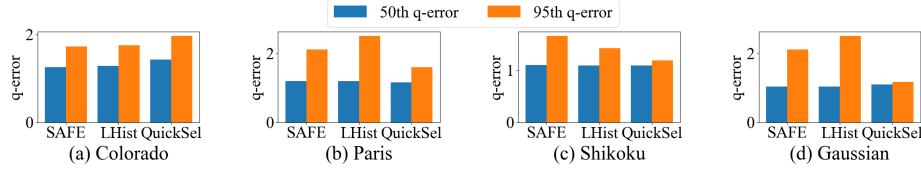


Fig. 7: Accuracy evaluation on different datasets

Therefore, m in SAFE was tuned accordingly. Note that this section evaluates the cardinality estimation models in the case where no updates occur.

Construction time. Fig. 6 shows the construction time of each method (best viewed in color). The construction of SAFE can be divided into three parts: data partitioning based on a quad-tree, sampling, and the training of regression models. If we assume that the quad-tree is built in advance (for efficiently running range queries), we can ignore the quad-tree construction time. Then, under this assumption, its construction time can be the fastest.

Accuracy. Fig. 7 shows the 50th and 95th q-errors of each method. SAFE and LHist show similar q-errors (but SAFE is usually better). QuickSel usually shows the lowest 95th q-error among the three methods. Nevertheless, (i) all three methods show similar 50th q-errors and (ii) they are close to 1 (nearly no error). We therefore see that SAFE can (often) yield an accurate cardinality for a given range query without any query workloads.

Accuracy vs. estimation time. We next investigate the trade-off relationship between accuracy and estimation time, and the result is depicted in Fig. 8. We see that QuickSel has the worst trade-off, whereas SAFE does the best. If applications require faster estimation time, the accuracy of QuickSel degrades quickly. On the other hand, SAFE does not lose its accuracy so much even when a faster estimation time (e.g., less than 10 microseconds) is required. This is a great advantage of SAFE.

4.3 Result: Dynamic Data Case

We turn our attention to the dynamic data case. This section evaluates accuracy and update time. We here note that estimation time does not change even when we update the models. As competitors, we used QuickSel and SAFE without

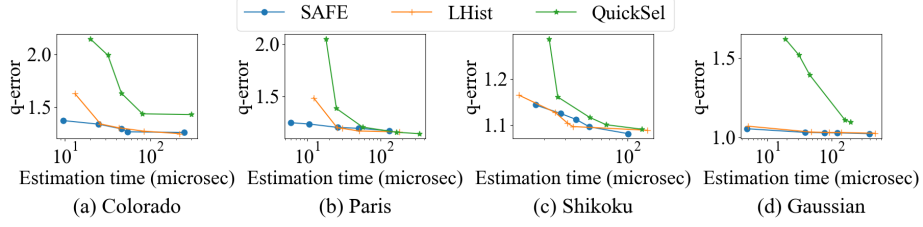


Fig. 8: Comparison evaluation on estimation time versus accuracy (50th q-error)

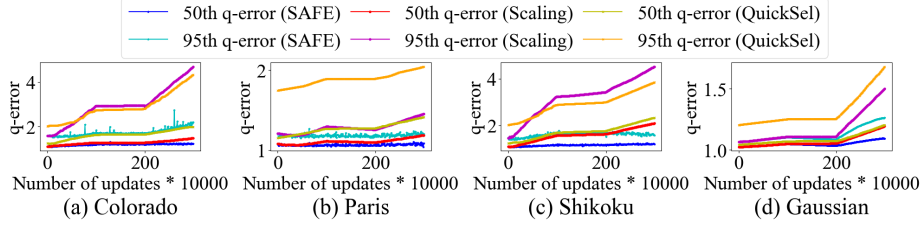


Fig. 9: Continuous update evaluation

Table 3: Average update time of SAFE per an update (microsec)

Dataset	Colorado	Paris	Shikoku	Gaussian
Update time	9.9	19.8	6.2	14.1

update⁴. This strategy, denoted by Scaling, scales the estimation results of the *non-updated* model by using the original and updated dataset sizes.

We sequentially updated the datasets by using one million insertions, one million deletions, and one million replacements. The insertions followed a Gaussian distribution within the corresponding data space, and the deletions are randomly selected from the original datasets. Each replacement replaced a randomly selected original point with a synthetic point following Gaussian distribution (i.e., the data distribution gradually changed). We used 300 queries with selectivity of 0.1 and 0.01 for the evaluation. We test for every 100 data updates. Fig. 9 shows the continuous update results on different datasets. SAFE keeps high accuracy as its 50th q-error keeps being almost 1.0 and the 95th q-error is also generally low. On the other hand, Scaling and QuickSel cannot keep accurate estimation. Their q-errors always become larger as receiving more updates.

We next investigated the update time of SAFE per an update, which is shown in Table 3. We see that the update time is around 10 microseconds and quite fast. This is a desirable result for frequent data updates. Given the experimental

⁴ Recall Section 2, and, to update QuickSel’s model for the up-to-date dataset, we need to compute the results of all existing queries, which requires more than a few seconds even for hundreds of queries. This is too slow for dynamic data, thereby we followed the original strategy of QuickSel (i.e., its model was not updated). In addition, LHist does not support data updates and needs reconstruction (incurring at least a few seconds for each update), thus we did not use it as a competitor.

results in Sections 4.2 and 4.3, it is clear that SAFE is better than the state-of-the-art cardinality estimation methods, since SAFE is better at estimation-accuracy trade-off and updatability than them.

5 Related Work

Data-driven cardinality estimation. Data-driven cardinality estimators learn the data distribution from a given dataset directly. (SAFE belongs to this category.) Latest learning-based methods tend to consider this task as an unsupervised problem. The Sum-Product network [8] recursively assigns points into clusters. It uses a probabilistic graphic model to approximate the joint data distribution. Autoregressive models [7, 21] use a neural network to approximate the joint distribution in a conditional way and estimate the cardinality without any independent assumptions. As mentioned earlier, neural network-based structures cannot be trained fast thus are not suitable for dynamic data.

Query-driven cardinality estimation. Many SOTA cardinality methods tend to employ this query-driven style [6], because deep learning models well suit this style. They can capture complex queries such as joins across multiple tables. QuickSel [15] is the most representative among non-deep learning technologies. However, its supervised nature, which requires queries to learn the queried data distribution, limits applications. Query-driven methods need query samples, and they are sensitive to query workload shifts. While QuickSel can adapt to such changes, the update cost is high, as mentioned in Section 4.

Learned spatial indices. Learned indices provide a new way to construct spatial indices [19]. Waffle [13] is introduced to handle highly dynamic datasets. It provides a novel bulk loading method to optimize the utilization of disk pages (which is a different assumption from ours). It can automatically optimize the index by managing the trade-off between the query and update speed. Waffle requires a query workload in advance, so its idea cannot be used in our setting.

6 Conclusion

We proposed SAFE for cardinality estimation on dynamic spatial data. SAFE employs (i) sampling-based learning for fast training, (ii) 2-tier regression models to enable fast and accurate estimation while having an efficient update ability, and (iii) an incremental model update strategy. Combining these three mechanisms achieved low latency and accurate estimation while being robust to data updates. Our experimental results demonstrate that SAFE outperforms the state-of-the-art estimators while supporting efficient model updates.

Acknowledgements

This work was partially supported by AIP Acceleration Research JPMJCR23U2, JST, and JSPS KAKENHI Grant Number 24K14961.

References

1. Amagata, D., Hara, T.: Monitoring maxrs in spatial data streams. In: EDBT. pp. 317–328 (2016)
2. Amagata, D., Hara, T.: Identifying the most interactive object in spatial databases. In: ICDE. pp. 1286–1297 (2019)
3. Bentley, J.L.: Multidimensional binary search trees used for associative searching. *Communications of the ACM* **18**(9), 509–517 (1975)
4. Eldawy, A., Mokbel, M.F.: The era of big spatial data. In: ICDE Workshops. pp. 42–49 (2015)
5. Finkel, R.A., Bentley, J.L.: Quad trees a data structure for retrieval on composite keys. *Acta informatica* **4**, 1–9 (1974)
6. Han, Y., Wu, Z., Wu, P., Zhu, R., Yang, J., Tan, L.W., Zeng, K., Cong, G., Qin, Y., Pfadler, A., Qian, Z., Zhou, J., Li, J., Cui, B.: Cardinality estimation in dbms: A comprehensive benchmark evaluation. *PVLDB* p. 752–765 (2021)
7. Hasan, S., Thirumuruganathan, S., Augustine, J., Koudas, N., Das, G.: Deep learning models for selectivity estimation of multi-attribute queries. In: SIGMOD. p. 1035–1050 (2020)
8. Hilprecht, B., Schmidt, A., Kulesa, M., Molina, A., Kersting, K., Binnig, C.: Deepdb: Learn from data, not from queries! *PVLDB* **13**(7), 992–1005 (2020)
9. Kipf, A., Kipf, T., Radke, B., Leis, V., Boncz, P.A., Kemper, A.: Learned cardinalities: Estimating correlated joins with deep learning. In: CIDR (2019)
10. Lesh, F.H.: Multi-dimensional least-squares polynomial curve fitting. *Communications of the ACM* **2**(9), 29–30 (1959)
11. Liu, Q., Shen, Y., Chen, L.: Lhist: Towards learning multi-dimensional histogram for massive spatial data. In: ICDE. pp. 1188–1199 (2021)
12. Meng, Z., Wu, P., Cong, G., Zhu, R., Ma, S.: Unsupervised selectivity estimation by integrating gaussian mixture models and an autoregressive model. In: EDBT. pp. 2–247 (2022)
13. Moti, M.H., Simatis, P., Papadias, D.: Waffle: A workload-aware and query-sensitive framework for disk-based spatial indexing. *PVLDB* **16**(4), 670–683 (2022)
14. Nishio, S., Amagata, D., Hara, T.: Lamps: Location-aware moving top-k pub/sub. *IEEE Transactions on Knowledge & Data Engineering* **34**(01), 352–364 (2022)
15. Park, Y., Zhong, S., Mozafari, B.: Quicksel: Quick selectivity learning with mixture models. In: SIGMOD. p. 1017–1033 (2020)
16. Sun, L., Li, C., Ji, T., Chen, H.: Mose: A monotonic selectivity estimator using learned cdf. *IEEE Transactions on Knowledge and Data Engineering* **35**(3), 2823–2836 (2023)
17. Tang, M., Yu, Y., Malluhi, Q.M., Ouzzani, M., Aref, W.G.: Locationspark: A distributed in-memory data management system for big spatial data. *PVLDB* **9**(13), 1565–1568 (2016)
18. Wang, J., Chai, C., Liu, J., Li, G.: Face: A normalizing flow based cardinality estimator. *PVLDB* **15**(1), 72–84 (2021)
19. Wu, J., Zhang, Y., Chen, S., Wang, J., Chen, Y., Xing, C.: Updatable learned index with precise positions. *PVLDB* **14**(8), 1276–1288 (2021)
20. Wu, P., Cong, G.: A unified deep model of learning from both data and queries for cardinality estimation. In: SIGMOD. pp. 2009–2022 (2021)
21. Yang, Z., Kamsetty, A., Luan, S., Liang, E., Duan, Y., Chen, X., Stoica, I.: Neurocard: One cardinality estimator for all tables. *PVLDB* **14**(1), 61–73 (2020)