An Experimental Study of Learned Cardinality Estimation

by

Xiaoying Wang

B.Sc., Tongji University, 2016

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2020

Declaration of Committee

Name:	**Xiaoying Wang**
Degree:	**Master of Science**
Thesis title:	**An Experimental Study of Learned Cardinality Estimation**
Committee:	**Chair:** Steven Bergner University Research Associate, Computing Science
	Jiannan Wang Supervisor Associate Professor, Computing Science
	Tianzheng Wang Committee Member Assistant Professor, Computing Science
	Ke Wang Examiner Professor, Computing Science

Abstract

Cardinality estimation is a fundamental but long unresolved problem in query optimization. Recently, multiple papers from different research groups consistently report that learned models have the potential to replace existing cardinality estimators. In this thesis, we ask a forward-thinking question: *Are we ready to deploy these learned cardinality models in production?* Our study consists of three main parts. Firstly, we focus on the static environment (i.e., no data updates) and compare five new learned methods with eight traditional methods on four real-world datasets under a unified workload setting. The results show that learned models are indeed more accurate than traditional methods, but they often suffer from high training and inference costs. Secondly, we explore whether these learned models are ready for dynamic environments (i.e., frequent data updates). We find that they cannot catch up with fast data updates and return large errors for different reasons. For less frequent updates, they can perform better but there is no clear winner among themselves. Thirdly, we take a deeper look into learned models and explore when they may go wrong. Our results show that the performance of learned methods can be greatly affected by the changes in correlation, skewness, or domain size. More importantly, their behaviors are much harder to interpret and often unpredictable. Based on these findings, we identify two promising research directions (control the cost of learned models and make learned models trustworthy) and suggest a number of research opportunities. We hope that our study can guide researchers and practitioners to work together to eventually push learned cardinality estimators into real database systems.

Keywords: Cardinality Estimation; Query Optimizer; Machine Learning

Acknowledgements

Firstly, I would like to express my sincerest appreciation to my senior supervisor, Prof. Jiannan Wang. His patience, guidance and insights in the field have made my master study an extremely inspiring and pleasant experience and I have learned a great deal from him in both scientific research and life in general.

My gratitude also goes to the rest of my thesis committee Prof. Tianzheng Wang, Prof. Ke Wang and Dr. Steven Bergner for their kindly serving and providing valuable feedback about this thesis. I also want to give my gratitude to Qingqing Zhou, for his constructive and insightful comments to this project.

Secondly, I would like to thank all the members from Database System Lab, especially Changbo Qu and Weiyuan Wu, who worked together with me on this work. I would not have been able to complete this research without your contribution and help.

Finally, to my parents and my husband Zijing, thanks for being supportive to every decision I make, without which I would not have been able to start my study in the beginning.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The rise of "ML for DB" has sparked a large body of exciting research studies exploring how to replace existing database components with learned models [39, 34, 41, 67, 84, 96]. Impressive results have been repeatedly reported from these papers, which suggest that "ML for DB" is a promising research area for the database community to explore. To maximize the impact of this research area, one natural question that we should keep asking ourselves is: *Are we ready to deploy these learned models in production?*

In this thesis, we seek to answer this question for cardinality estimation. In particular, we focus on *single-table cardinality estimation*, a fundamental and long standing problem in query optimization [93, 18]. It is the task of estimating the number of tuples of a table that satisfy the query predicates. Database systems use a query optimizer to choose an execution plan with the estimated minimum cost. The performance of a query optimizer largely depends on the quality of cardinality estimation. A query plan based on a wrongly estimated cardinality can be orders of magnitude slower than the best plan [44].

Multiple recent papers [93, 36, 18, 30, 32] have shown that learned models can greatly improve the cardinality estimation accuracy compared with traditional methods. However, their experiments have a number of limitations (see Section 2.1.5 for more detailed discussion). Firstly, they do not include all the learned methods in their evaluation. Secondly, they do not use the same datasets and workload. Thirdly, they do not extensively test how well learned methods perform in dynamic environments (e.g., by varying update rate). Lastly, they mainly focus on when learned methods will go right rather than when they may go wrong.

We overcome these limitations and conduct comprehensive experiments and analyses. The thesis makes four contributions:

Are Learned Methods Ready For Static Environments? We propose a unified workload generator and collect four real-world benchmark datasets. We compare five new learned methods with eight traditional methods using the same datasets and workload in static environments (i.e., no data updates). The results on accuracy are quite promising. In terms of training/inference time, there is only one method [18] that can achieve similar perfor-

mance with existing DBMSs. The other learned methods typically require $10-1000\times$ more time in training and inference. Moreover, all learned methods have an extra cost for hyper-parameter tuning.

Are Learned Methods Ready For Dynamic Environments? We explore how each learned method performs by varying update rate on four real-world datasets. The results show that learned methods fail to catch up with fast data updates and tend to return large error for various reasons (e.g., the stale model processes too many queries, the update period is not long enough to get a good updated model). When data updates are less frequent, learned methods can perform better but there is no clear winner among themselves. We further explore the update time vs. accuracy trade-off, and investigate how much GPU can help learned methods in dynamic environments.

When Do Learned Methods Go Wrong? We vary correlation, skewness, and domain size, respectively, on a synthetic dataset, and try to understand when learned methods may go wrong. We find that all learned methods tend to output larger error on more correlated data, but they react differently w.r.t. skewness and domain size. Due to the use of black-box models, their wrong behaviors are very hard to interpret. We further investigate whether their behaviors follow some simple and intuitive logical rules. Unfortunately, most of them violate these rules. We discuss four issues related to deploying (black-box and illogical) learned models in production.

Research Opportunities. We identify two future research directions: i) control the cost of learned methods and ii) make learned methods trustworthy, and suggest a number of promising research opportunities. We hope this work can attract more research efforts in these directions and eventually overcome the barriers of deploying learned estimators in production.

The rest of the thesis is organized as follows: We present a survey on learned cardinality estimation as well as its related works in Chapter 2 and describe the general experimental setup in Chapter 3. We explore whether learned methods are ready for static environments in Chapter 4 and for dynamic environments in Chapter 5, and examine when learned methods go wrong in Chapter 6. We present our conclusions in Chapter 7. Finally, future works and research opportunities are discussed in Chapter 8.

Chapter 2

Background

2.1 Learned Cardinality Estimation

In this section, we first formulate the cardinality estimation problem, then put new learned methods into a taxonomy and present how each method works, and finally discuss the limitations of existing evaluation on learned methods.

2.1.1 Problem Statement

Consider a relation R with n attributes $\{A_1, \ldots, A_n\}$ and a query over R with a conjunctive of d predicates:

```
SELECT COUNT(*) FROM R
WHERE θ1 AND ··· and θd,
```

where θ_i ($i \in [1, d]$) can be an *equality predicate* like $A = a$, an *open range predicate* like $A \leq a$, or a *close range predicate* like $a \leq A \leq b$. The goal of cardinality estimation is to estimate the answer to this query, i.e., the number of tuples in R that satisfy the query predicates. An equivalent problem is called *selectivity estimation*, which computes the *percentage* of tuples that satisfy the query predicates.

Table 2.1: Taxonomy of New Learned Cardinality Estimators.

	Methodology	Input	Model
MSCN [36]	Regression	Query+Data	Neural Network
LW-XGB [18]	Regression	Query+Data	Gradient Boosted Tree
LW-NN [18]	Regression	Query+Data	Neural Network
DQM-Q [30]	Regression	Query	Neural Network
Naru [93]	Joint Distribution	Data	Autoregressive Model
DeepDB [32]	Joint Distribution	Data	Sum Product Network
DQM-D [30]	Joint Distribution	Data	Autoregressive Model

2.1.2 Taxonomy

The idea of using ML for CE is not new (see Section 2.2 for more related work). The novelty of recent learned methods is to adopt more advanced ML models, such as deep neural networks [36, 18, 30], gradient boosted trees [18], sum-product networks [32], and deep autoregressive models [93, 30]. We call these methods "new learned methods" or omit new, i.e., "learned methods", if the context is clear. In contrast, we refer to "traditional methods" as the methods based on histogram or classic ML models like KDE and Bayesian Network.

Table 2.1 shows a taxonomy of new learned methods[1]. Based on the methodology, we split them into two groups - *Regression* and *Joint Distribution* methods. *Regression* methods (a.k.a *query-driven* methods) model CE as a regression problem and aim to build a mapping between queries and the CE results via feature vectors, i.e., $query \rightarrow feature\_vector \rightarrow CE\_result$. *Joint Distribution* methods (a.k.a *data-driven* methods) model CE as a joint probability distribution estimation problem and aim to construct the joint distribution from the table, i.e., $P(A_1, A_2, \cdots, A_n)$, then estimate the cardinality. The Input column represents what is the input to construct each model. Regression methods all require queries as input while joint distribution methods only depend on data. The Model column indicates which type of model is used correspondingly. We will introduce these methods in the following.

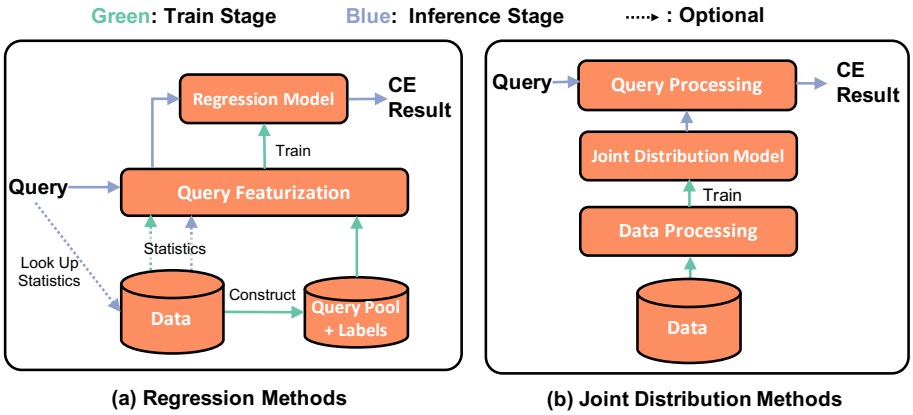


Figure 2.1: Workflow of Learned Methods.

[1] Naru, DeepDB and MSCN are named by their authors. For convenience of discussion, we give others the following short names. Lightweight Gradient Boosting Tree (LW-XGB) and Lightweight Neural Network (LW-NN) are two models from [18]. From [30], two complementary methods are proposed, Data&Query Model - Data (DQM-D) and Data&Query Model - Query (DQM-Q).

2.1.3 Methodology 1: Regression

Workflow. Figure 2.1(a) depicts the workflow of regression methods. In the *training stage*, it first constructs a query pool and gets the label (CE result) of each query. Then, it goes through the query featurization module, which converts each query to a feature vector. The feature vector does not only contain query information but also optionally include some statistics (like a small sample) from the data. Finally, a regression model is trained on a set of ⟨feature vector, label⟩ pairs. In the *inference stage*, given a query, it converts the query to a feature vector using the same process as the training stage, and applies the regression model to the feature vector to get the CE result. To handle *data updates*, regression methods need to update the query pool and labels, generate new feature vectors, and update the regression model. In order to reduce the cost, sometimes we can be collect labels using a sample of the data instead of running queries on the whole dataset.

There are four regression methods: MSCN, LW-XGB, LW-NN, and DQM-Q. One common design choice in them is the usage of log-transformation on the selectivity label since the selectivity often follows a skewed distribution and log-transformation is commonly used to handle this issue [19]. These works vary from many perspectives, such as their input information, query featurization, and model architecture.

MSCN [36] introduces a specialized deep neural network model termed multi-set convolutional network (MSCN). MSCN can support join cardinality estimation. It represents a query as a feature vector which contains three modules (i.e., table, join, and predicate modules). Each module is a two-layer neural network and different module outputs are concatenated and fed into a final output network, which is also a two-layer neural network. MSCN enriches the training data with a materialized sample. A predicate will be evaluated on a sample, and a bitmap, where each bit indicates whether a tuple in the sample satisfies the predicate or not, will be added to the feature vector. This enrichment has been proved to make obvious positive impact on the model performance [36, 93].

LW-XGB/NN [18] introduces a lightweight selectivity estimation method. Its feature vector consists of two parts: range features + CE features. The range features represent a set of range predicates: $\langle a_1, b_1, a_2, b_2, \cdots, a_n, b_n \rangle$. The CE features represent heuristic estimators (e.g., the one that assumes all columns are independent). Note that the CE features can be cheaply derived from the statistics available in the database system. LW-NN (LW-XGB) train a neural network (gradient boost tree) model using the generated features. Unlike MSCN which minimizes the mean q-error, they minimize the mean square error (MSE) of the log-transformed label, which equals to minimizing the geometric mean of q-error with more weights on larger errors and also can be computed efficiently.

DQM-Q [30] proposes a different featurization approach. It uses one-hot encoding to encode categorical columns and treats numerical attributes as categorical attributes by automatic

discretization [15]. DQM-Q trains a neural network model. When a real-world query workload is available, DQM-Q is able to augment the training set and train the model with the augmented set.

2.1.4 Methodology 2: Joint Distribution

Workflow. Figure 2.1(b) depicts the workflow of joint distribution methods. In the *training stage*, it transforms the data into a format ready for training a joint distribution model. In the *inference stage*, given a query, it generates one or multiple requests to the model and combine the model inference results into the final CE result. To handle *data updates*, joint distribution methods need to update or retrain the joint distribution model.

There are three joint distribution methods: Naru, DeepDB, and DQM-D. Compared to traditional methods like histogram and sampling, these new methods adopt more complex models to further capture additional information in the data, such as fine-grained correlation or conditional probability between columns.

Autoregressive Model. Naru [93] and DQM-D [30] propose similar ideas. They factorize the joint distribution into conditional distributions using the product rule:

$$P(A_1, A_2, ..., A_n) = P(A_1)P(A_2|A_1) \cdots P(A_n|A_1, ..., A_{n-1})$$

They adopt the state-of-the-art deep autoregressive models such as MADE [25] and Transformer [88] to approximate the joint distribution and achieve an impressive estimation accuracy.

The joint distribution can directly return results to point queries. To support range queries, they adopt a sampling based method, which runs importance sampling in an adaptive fashion. Specifically, Naru uses a novel approximation technique named progressive sampling, which samples values column by column according to each internal output of conditional probability distribution. DQM-D adopts an algorithm [46] originally designed for Monte-Carlo multi-dimensional integration, which conducts multiple stages of sampling. At each stage, it selects sample points in proportion to the contribution they make to the query cardinality according to the result from the previous stage.

Sum-Product Network. DeepDB [32] builds Sum-Product Networks (SPNs) [70] to capture the joint distribution. The key idea is to recursively split the table into different clusters of rows (creating a sum node to combine them) or clusters of columns (assuming different column clusters are independent and creating a product node to combine them). KMeans is used to cluster rows and Randomized Dependency Coefficients [51] is used to identify independent columns. Leaf nodes in an SPN represent a single attribute distribution, which can be approximated by histograms for discrete attributes or piecewise linear functions for continuous attributes.

2.1.5 Limitations of Existing Experiments

As pointed in the Introduction, existing experimental studies have a number of limitations. We provide more detail in this section.

Firstly, many new learned methods have not been compared with each other directly. Figure 2.2 visualizes the available comparison results using a directed graph. Each node represents a method, and if method A has compared with method B in A's paper, we draw a directed edge from A to B. Since many methods were proposed in the same year or very close period, the graph is quite sparse and misses over half of the edges. For example, LW-XGB/NN is one of the best regression methods, but it has no edge with any other method. DeepDB and Naru are two state-of-the-art joint distribution methods, but there is no edge between them.

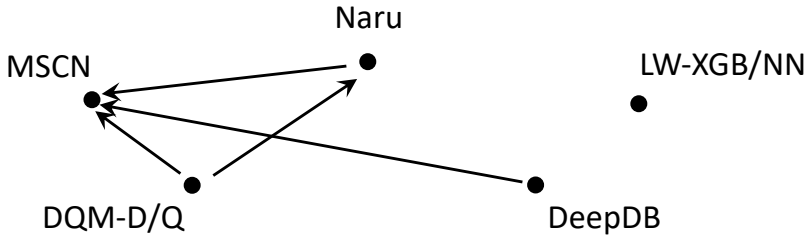


Figure 2.2: Comparison results available in existing studies.

Secondly, there is no standard about which datasets to use and how to generate workloads. Other than the IMDB dataset (adopted by MSCN and DeepDB), none of the datasets adopted in one work appear in another work. As for workloads, these works generate synthetic queries differently. Table 2.2 compares their generated workloads. For join queries in the JOB-light benchmark (used in MSCN and DeepDB), we report their properties related to single table. $|D|$ denotes the number of columns in the dataset and OOD (out-of-domain) means that the predicates of a query are generated independently. Such queries often lead to zero cardinality.

Table 2.2: Workload used in existing experimental studies.

	Predicate Number	**Operator**		**Consider OOD**
		Equal	**Range**	
MSCN	$0 \sim \|D\|$	✓	✓	×
LW-XGB/NN	$2 \sim \|D\|$	×	close range	✓
Naru	$5 \sim 11$	✓	open range	✓
DeepDB	$1 \sim 5$	✓	✓	×
DQM-D/Q	$1 \sim \|D\|$	✓	×	✓
Our Workload	$1 \sim \|D\|$	✓	✓	✓

Thirdly, existing works are mostly focused on the static environment (i.e., no data update setting). However, dynamic environments are also common in practice. Some papers have

explored how their method performs when the data updates, but the way that they update the data varies. As a result, the performance numbers cannot be used to compare between methods. Furthermore, existing studies have not extensively explored the trade-off between accuracy and updating time. For example, Naru is a more accurate method but requires longer time to update the model. It is unclear whether it can still give good accuracy for high update rates.

2.2 Related Work

Here we introduce some related works to learned cardinality estimation.

Single Table Cardinality Estimation. Histogram is the most common cardinality estimation approach and has been studied extensively [69, 72, 33, 60, 23, 71, 27, 28, 85, 1, 6, 48, 80, 56, 59] and adopted in database products. Sampling based methods [49, 91, 24, 75, 95] have the advantage to support more complex predicates than range predicates. Prior work mainly adopts traditional machine learning techniques to estimate cardinality, such as curve-fitting [9], wavelet [57], KDE [31], uniform mixture model [64], and graphical models [26, 14, 87]. Early works [43, 52, 50, 2] also use neural network models to approximate the data distribution in a regression fashion. In comparison, new learned methods have shown more promising results [36, 18].

Join Cardinality Estimation. Traditional database systems estimate the cardinality of joins following simple assumptions such as uniformity and independence [44]. Some works [36, 32] can support joins directly, while others [35, 89, 17, 92] study how to extend single table cardinality estimation methods to support join queries. Empirical study [63] evaluates different deep learning architectures and machine learning models on select-project-join workloads. Leis et. al [45] propose an index-based sampling technique which is cheap but effective. Focusing on a small amount of "difficult" queries, some works [90, 68] introduce a re-optimization procedure during inference to "catch" and correct the large errors, while another line of research tries to avoid poor plans by inferring the upper bound of the intermediate join cardinality [7].

End-to-End Query Optimization. Recently, more and more works try to tackle the query optimization problem in an end-to-end fashion. Sun et. al [81] propose a learning-based cost estimation framework based on a tree-structured model, which estimate both cost and cardinality simultaneously. Pioneer work [62] shows the possibility of learning state representation of query optimization for the join tree with reinforcement learning, and many follow-up works [54, 40, 86, 94] reveal the effectiveness of using deep reinforcement learning for join order selection. Marcus et. al propose Neo [55], which uses deep learning to generate query plans directly. There are also several end-to-end query optimization systems [3, 97, 78] available in the open-source community.

Benchmark and Empirical Study in Cardinality Estimation. Leis et. al [44] propose the Join Order Benchmark (JOB), which is based on the real-world IMDB dataset with synthetic queries having 3 to 16 joins [44]. Unlike JOB, we focus on single table cardinality estimation. Ortiz et. al [63] provide an empirical analysis on the accuracy, space and time trade-off across several deep learning and machine learning model architectures. Our study is different from their work in many aspects. We include both data-driven and query-driven learned methods (whereas they focus on query-driven models) and both static and dynamic settings. Also we try to explore when learned models would go wrong with controlled synthetic datasets and propose simple logical rules to evaluate them. Harmouch et. al [29] conduct an experimental survey on cardinality estimation, but their target is on estimating the number of distinct values, which is different from this thesis.

Machine Learning for Database Systems. Zhou et. al [98] provide a thorough survey on how ML and DB can benefit each other. In addition to cardinality estimation, ML has the potential to replace and enhance other components in database systems such as indexes [39] and sorting algorithms [41]. Another aspect is to leverage ML to automate database configurations like knob tuning [96, 84], index selection [67], and view materialization [34].

Chapter 3

Experimental Setup

Our study evaluates learned cardinality estimators under different settings. We describe the general setup used in all of our experiments in this chapter.

Evaluation Metric. We use q-error as our accuracy metric to measure the quality of the estimation result. Q-error is a symmetric metric which computes the factor by which an estimate differs from the actual cardinality:

$$error = \frac{max(est(q), act(q))}{min(est(q), act(q))}.$$

For example, if a query's actual cardinality is 10 and estimated cardinality is 100, then $error = \frac{max(100,10)}{min(100,10)} = 10$.

Q-error is the metric adopted by all learned methods [36, 18, 93, 32, 30]. It measures the relative error, which can penalize large and small results to the same extent. Furthermore, it has been proved to be directly related to the plan quality in query optimization [58].

Learned Methods & Implementation. As shown in Table 2.1, there are five recently published papers on learned cardinality estimation: Naru [93], MSCN [36], LW-XGB/NN [18], DeepDB [32], and DQM [30]. We exclude DQM from our study since its data driven model has a similar performance with Naru and its query driven model does not support our workload (confirmed with DQM's authors).

For Naru and DeepDB, we adopt the implementation released by the authors with minor modifications in order to support our experiments. We choose ResMADE as basic autoregressive building block for Naru because it is both efficient and accurate. For MSCN, since the original model supports join query, it needs extra input features to indicate different joins and predicates on different tables. To ensure a fair comparison on single table cardinality estimation, we modify the original code by only keeping features represent predicates and qualifying samples. We implement both neural network (LW-NN, on PyTorch [65]) and gradient boosted tree (LW-XGB, on XGBoost [10]) approach for LW-XGB/NN according to the

description in its original paper [18], and use Postgres's estimation result on single column to compute the CE features.

Hardware and Platform. We perform our experiments on a server with 16 Intel Xeon E7-4830 v4 CPUs (2.00GHz). For the neural network models (Naru, MSCN, LW-NN), we run them not only on CPU but also on a NVIDIA Tesla P100 GPU to gain more insights under different settings.

Our Study Questions. Our study is driven by the question: are we ready for learned cardinality estimators? In order to answer this, we evaluate learned cardinality methods under both static (Chapter 4) and dynamic (Chapter 5) settings. In order to gain more insights, we further examine the situations when learned methods do not perform well (Chapter 6).

Chapter 4

Are Learned Methods Ready for Static Environments?

Are learned estimators more accurate than traditional methods in static environment? What is the cost for the high accuracy? In this chapter, we first compare the accuracy of learned methods with traditional methods, and then measure their training and inference time in order to see whether they are ready for production.

4.1 Setup

Dataset. We use four real-world datasets with various characteristics (Table 4.1). We choose these datasets because first, the size of these datasets are in different magnitudes and the ratio between categorical and numerical columns varies; second, each dataset has been used in the evaluation of at least one prior work in this field.

Table 4.1: Dataset characteristics. “Cols/Cat" means the number of columns and categorical columns; “Domain" is the product of the number of distinct values for each column.

Dataset	Size(MB)	Rows	Cols/Cat	Domain
Census	4.8	49K	13/8	10^{16}
Forest	44.3	581K	10/0	10^{27}
Power	110.8	2.1M	7/0	10^{17}
DMV	972.8	11.6M	11/10	10^{15}

1. Census [16]: Also known as the “Adult” dataset, which is extracted from the 1994 Census database. We remove the column fnlwgt since its values are nearly identical and thus cause the cardinality result to be either 0 or 1 whenever a predicate is placed on it.

2. Forest [16]: Forest cover type dataset consists of 54 attributes. As in [18], we keep the first 10 numerical columns for evaluation (since the rest of the attributes are binary).

3. Power [16]: Household electric power consumption data gathered in 47 months. The same with [18] we use the 7 measurement attributes in our evaluation.

4. DMV [61]: Vehicle, snowmobile and boat registration records from the State of New York. We directly adopt the same snapshot, which contains 11,591,877 tuples and 11 attributes, from previous work [93].

Workload. We describe our unified workload generator. The goal of our workload generator is to be able to cover all the workload settings used in existing learned methods (see Table 2.2).

Intuitively, a query with d predicates can be thought of as a hyper-rectangle in a d-dimensional space. A hyper-rectangle is controlled by its center and width. Correspondingly, a query is controlled by its *query center* and *range width*. For example, consider a query with $d = 2$ predicates:

```
SELECT COUNT(*) FROM R
WHERE 0 ≤ A1 ≤ 20 AND 20 ≤ A2 ≤ 100
```

Its query center is $(\frac{20-0}{2}, \frac{100-20}{2}) = (10, 40)$ and its range width is $(20 - 0,\ 100 - 20) = (20, 80)$.

There are two ways to generate query centers. For ease of illustration, suppose that we want to generate a query center for columns A_1, A_2. The first way (①) is to randomly select a tuple t from the table. Let $t[A_1], t[A_2]$ denote the attribute values of the tuple on A_1 and A_2. Then, we set the query center to $(t[A_1], t[A_2])$. The second way (②) is to independently draw a random value c_1 and c_2 from the domain of A_1 and A_2, respectively, and set the query center to (c_1, c_2). ② is called out-of-domain (OOD in Table 2.2), which aims to test the robustness of learned estimators more comprehensively from the entire joint domain.

There are two ways to generate range widths. Let the domain for A_i be $[\min_i, \max_i]$ and the domain size be $\text{size}_i = \max_i - \min_i$. The first way (❶) is to uniformly select a value w_i from $[0, \text{size}_i]$. The second way (❷) is to select a value from an exponential distribution with a parameter λ_i (we set $\lambda = 10/\text{size}_i$ by default). Note that if A_i is a categorical column, we will only generate an equality predicate for it, thus the width is set to zero in this case. If a range on one side is larger than $\max_i$ or smaller than $\min_i$, then it becomes an open range query. Thus, our workload contains both open and close range queries.

Our workload generator covers all the above settings (①, ②, ❶, ❷). To generate a query, we first uniformly select a number d from 1 to $|D|$ and randomly sample d distinct columns to place the predicates. The query center is generated from ① and ② with a probability of 90% and 10%, respectively, and the range width is generated from ❶ and ❷ in equal proportions. The reason that we do not use an equal probability for the query center is that OOD is typically less common than the other way in real workloads. Figure 4.1 shows

the selectivity distribution of generated workloads on different datasets, which results in a broad spectrum.

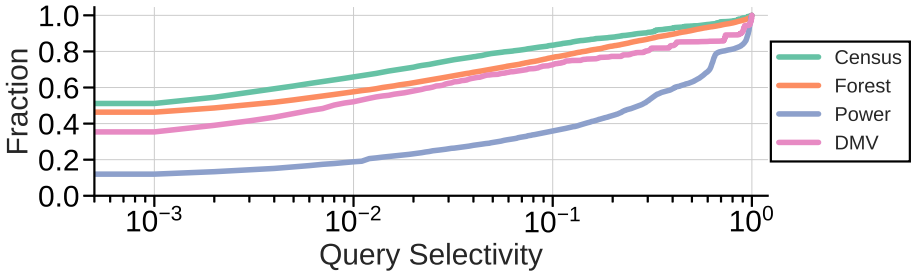


Figure 4.1: Distribution of workload selectivity.

Hyper-parameter Tuning. We describe hyper-parameter tuning for each model.

For neural network methods (Naru, MSCN, LW-NN), we control the model size within 1.5% of the data size for each dataset. For each method, we select four model architectures with different numbers of layers, hidden units, embedding size, etc. and train each model in different batch size and learning rate in accordance with the original papers. Since MSCN and LW-NN are query-driven methods, we select 10K queries as a validation set to determine which hyper-parameters are better. Since Naru is a data-driven method (i.e., no query as input), we use training loss to find optimal hyper-parameters.

For LW-XGB, we vary the number of trees (16, 32, 64...) as in [18]. Since LW-XGB is a query-driven method, similar to MSCN and LW-NN, we select 10K validation queries for it.

For DeepDB, we do a grid search on RDC threshold and minimum instance slice and only keep the models within the size budget (i.e., 1.5% of the data size). An interesting finding is that DeepDB does not output the training loss like Naru during construction, thus queries are needed for hyper-parameter tuning. However, DeepDB is designed to be a data-driven method, which is not supposed to use queries. To ensure a fair comparison with other methods, we select a very small number of validation queries (i.e., 100 queries) for DeepDB to do hyper-parameter tuning.

To ensure a fair comparison, we use 100K queries to train all the query-driven methods (MSCN, LW-XGB/NN).

Traditional Techniques. We compare with a variety of traditional techniques, which are either used by real database systems or reported to achieve the state-of-the-art performance recently.

- *Postgres, MySQL and DBMS-A* are used to represent the performance of real database systems. We use PostgreSQL 11.5 and 8.0.21 MySQL Community Server in our experiment, and DBMS-A is a leading commercial database system. They estimate cardinality rapidly with simple statistics and assumptions. In order to let them achieve

their best accuracy level, we set the statistics target to the upper limit (10,000 for Postgres, 1024 for MySQL). For DBMS-A, we create several multi-column statistics in order to cover all columns with histograms.

- *Sampling* uses a uniform random sample to estimate the cardinality. We sample 1.5% tuples from each dataset to make the size budget the same as the learned models.

- *MHIST* [71] builds a multi-dimensional histogram on the entire dataset. We choose Maxdiff as the partition constraint with Value and Area being the sort and source parameter since it is the most accurate choice according to [72]. We run the MHIST-2 algorithm iteratively until it reaches to 1.5% of the data size.

- *QuickSel* [64] represents query-driven multi-dimensional synopsis approaches' performance. It models the data distribution with uniform mixture model by leveraging query feedback. We choose QuickSel because it shows better accuracy than query-driven histograms including STHoles [6] and ISOMER [80] in [64]. We use 10K queries to train the model.

- *Bayes* [13] shows the estimation results of probabilistic graphical model approaches [26, 87, 14]. We adopt the same implementation in [93], which uses progressive sampling to estimate range queries and shows a very promising accuracy.

- *KDE-FB* [31] represents the performance of modeling data distribution with kernel density models. It improves naive KDE by optimizing the bandwidth with query feedback. We sample 1.5% tuples from each dataset (max to 150K) and use 1K queries to train the model.

4.2 Are Learned Methods More Accurate?

We test all the methods using 10K queries on each dataset. Table 4.2 shows the q-error comparison result. Bold values in the "Traditional Methods" section denotes the minimum q-error that traditional methods can reach, while in the "Learned Methods" section it highlights the learned methods that can achieve a smaller (or equal) q-error than the best traditional method. The last row summaries the comparison by using "win" to denote learned methods beating traditional methods, and "lose" means the opposite.

Overall, learned methods are more accurate than traditional methods in almost all the scenarios. The best learned method can beat the best traditional method up to 14× on max q-error. The improvement over the three real database systems is particularly impressive. For example, they achieve 28×, 51×, 938×, and 1758× better max q-error on Census, Forest, Power and DMV, respectively. Even in the only exception that learned methods lose (50th on Forest), they can still achieve very similar performance to the best traditional result.

Table 4.2: Estimation errors on four real-world datasets.

Estimator	Census				Forest				Power				DMV			
	50th	95th	99th	Max	50th	95th	99th	Max	50th	95th	99th	Max	50th	95th	99th	Max
Traditional Methods																
Postgres	1.40	18.6	58.0	1635	1.21	17.0	71.0	9374	1.06	15.0	235	$2 \cdot 10^5$	1.19	78.0	3255	$1 \cdot 10^5$
MySQL	1.40	19.2	63.0	1617	1.20	48.0	262	7786	1.09	26.0	2481	$2 \cdot 10^5$	1.40	1494	$3 \cdot 10^4$	$4 \cdot 10^5$
DBMS-A	4.16	122	307	2246	3.44	363	1179	$4 \cdot 10^4$	1.06	8.08	69.2	$2 \cdot 10^5$	1.46	23.0	185	$3 \cdot 10^4$
Sampling	1.16	31.0	90.0	389	**1.04**	17.0	67.0	416	**1.01**	**1.22**	**8.00**	280	**1.01**	**1.42**	19.0	**231**
MHIST	4.25	138	384	1673	3.83	66.5	288	$2 \cdot 10^4$	4.46	184	771	$1 \cdot 10^5$	1.58	13.8	90.8	$3 \cdot 10^4$
QuickSel	3.02	209	955	6523	1.38	15.0	142	7814	3.13	248	$1 \cdot 10^4$	$4 \cdot 10^5$	126	$1 \cdot 10^5$	$4 \cdot 10^5$	$4 \cdot 10^6$
Bayes	**1.12**	**3.50**	**8.00**	303	1.13	7.00	29.0	1218	1.03	2.40	15.0	$3 \cdot 10^4$	1.03	1.85	**12.9**	$1 \cdot 10^5$
KDE-FB	1.18	23.0	75.0	**293**	**1.04**	**5.00**	**17.0**	**165**	**1.01**	1.25	9.00	**254**	**1.01**	1.50	36.0	283
Learned Methods																
MSCN	1.38	7.22	15.5	**88.0**	1.14	7.62	20.6	377	**1.01**	2.00	9.91	**199**	1.02	5.30	25.0	351
LW-XGB	1.16	**3.00**	**6.00**	594	1.10	**3.00**	**7.00**	220	1.02	1.72	**5.04**	5850	**1.00**	1.68	**6.22**	$3 \cdot 10^4$
LW-NN	1.17	**3.00**	**6.00**	829	1.13	**3.10**	**7.00**	1370	1.06	1.88	**4.89**	$4 \cdot 10^4$	1.16	3.29	22.1	$3 \cdot 10^4$
Naru	**1.09**	**2.50**	**4.00**	**57**	1.06	**3.30**	**9.00**	**153**	**1.01**	**1.14**	**1.96**	**161**	**1.01**	**1.09**	**1.35**	**16.0**
DeepDB	**1.11**	4.00	8.50	**59.0**	1.06	**5.00**	**14.0**	1293	1.00	1.30	**2.40**	1568	1.02	1.86	**5.88**	5086
L v.s. T	**win**	**win**	**win**	**win**	lose	**win**	**win**	**win**	**win**	**win**	**win**	**win**	**win**	**win**	**win**	**win**

Among all learned methods, Naru is the most robust and accurate one. It basically has the best q-error across all scenarios and keeps its max q-error within 200. As for query-driven methods, LW-XGB can achieve the smallest q-error in most situations except for max q-error, in which it cannot beat MSCN. We find that the queries which have large errors on LW-XGB and LW-NN usually follow the same pattern: the selectivity on each single predicate is large while the conjunctive of multiple such predicates is very small. This pattern cannot be well captured by the CE features (AVI, MinSel, EBO) adopted LW-XGB/NN. In comparison, MSCN can handle this situation better which may be due to the sample used in its input.

4.3 What Is the Cost For High Accuracy?

Since learned methods can beat the cardinality estimators used in real database systems by a large margin, can we just directly deploy them? In this section, we examine the cost of these highly accurate learned methods. We compare learned methods with database systems in terms of training time and inference time. Figure 4.2 shows the comparison result.

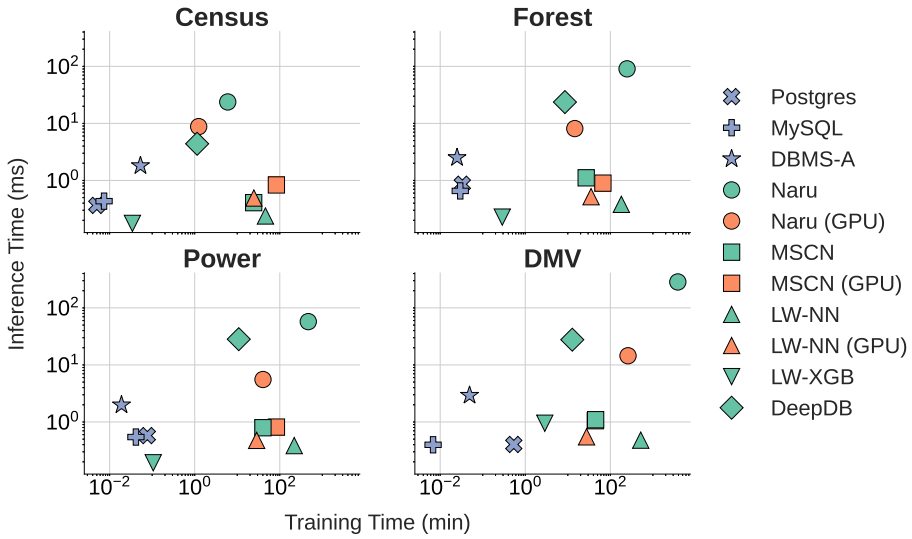


Figure 4.2: Training and inference time comparison between learned methods and real database system (MSCN's CPU and GPU results on DMV are overlapped).

Training Time. For learned methods, we record the time used to train the models reported in Table 4.2. For database systems, we record the time to run the statistics collection commands.

Database systems can finish constructing statistics in seconds on all datasets, while learned methods generally need minutes or even hours to train a model. LW-XGB tends to be the fastest learned methods, which can even achieve better performance than some database systems on small datasets since fewer trees are created. DeepDB is the second fastest and needs a few minutes to train the model. Since we run the same number of

epochs on all datasets, Naru's training time highly depends on the data size. With GPU, it only needs 1 minute on Census but takes more than 4 hours on DMV, and this time would be 5× to 15× slower on CPU. LW-NN also benefits from GPU acceleration, which takes around 30 minutes to finish training on all datasets but the time can be up to 20× longer if using CPU. On the other hand, MSCN exhibits similar training time on the two devices, and GPU is even 3.5× slower than CPU on small datasets. Our guess is that MSCN uses for loops to handle the conditional workflow when minimizing the mean q-error, which cannot make use of the strength of GPU and the overhead becomes more obvious when the model itself is small.

There is a tradeoff between training time and model accuracy. Neural network methods (Naru, MSCN and LW-NN) trained in an iterative fashion would produce larger error with fewer training iterations. For all these models, we adopt the same epochs reported in the original paper on all datasets, although some models can achieve similar performance with much fewer iterations. For example, using 80% less time, we can train a Naru model on DMV dataset with only slightly performance degrade. However, even if we only run 1 epoch on GPU, it will still be much slower than database systems. We will further explore this trade-off in Section 5.3.

Inference Time. We compute the average inference time of the 10K test queries by issuing the queries one by one. Figure 4.2 shows the result. For database systems, we approximate the time by the latency they return execution plan (without executing it), which should be longer than the real cardinality estimation time due to other overheads such as parsing and binding. Despite of that, all three DBMSs can finish the whole process in 1 or 2 milliseconds. Query-driven models (MSCN and LW-XGB/NN) are very competitive, which can achieve similar or better latency (but notice that DBMS's result includes other overheads). The remaining models are much slower. DeepDB needs around 25ms on the three larger datasets and takes an average of 5ms on Census. Naru's inference time is sensitive to the running device, which needs 5ms to 15 ms on GPU and CPU can be up to 20× slower.

The cardinality estimator could be invoked many times during query optimization. Long inference latency can be a blocking issue of bring these accurate learned estimators like Naru and DeepDB into production, especially for OLTP applications with short-running queries. In addition, shortening the inference time of these methods is not a trivial task. Take Naru as an example. Its bottleneck is the dependency of the selectivity computation for each attribute in the progressive sampling procedure, which needs to be done sequentially.

Hyper-parameter Tuning. Hyper-parameter tuning is another cost for learned methods. The learned models shown in Table 4.2 represent the models with the best hyper-parameters. Without hyper-parameter tuning, the learned models could perform very badly. Table 4.3 shows the ratio between the largest and the smallest max q-error among all the neural

network models trained during hyper-parameter tuning. We can see that the ratio for Naru, MSCN and LW-NN can be up to 10^5, 10^2 and 10, respectively.

Table 4.3: Ratio between the worst and best max q-error

Estimator	Census	Forest	Power	DMV
Naru	10.51	5.69	12.74	$4 \cdot 10^5$
MSCN	4.48	36.52	88.89	7.55
LW-NN	3.48	4.64	8.58	8.06

While essential for high accuracy, hyper-parameter tuning is a highly expensive process since it needs to train multiple models in order to find the best hyper-parameters. For example, as shown in Figure 4.2, Naru spends more than 4 hours in training a single model on DMV with GPU. If five models are trained, then Naru needs to spend 20+ hours (almost a day) on hyper-parameter tuning.

4.4 Main Findings

Our main findings of this chapter are summarized as follows:

- In our experiment, new learned estimators can deliver more accurate prediction than traditional methods in general and among learned methods, Naru shows the most robust performance.
- In terms of training time, new learned methods can be slower than DBMS products in magnitudes except for LW-XGB, which can achieve similar performance with database systems on small datasets.
- New learned estimators that based on regression models (MSCN and LW-XGB/NN) can be competitive to database systems in inference time, while methods that model the joint distribution directly upon data (Naru and DeepDB) requires much longer time.
- GPU can speed up the training and inference time of some of the new learned estimators, however it cannot make them as quick as DBMS products and sometimes introduce overhead.
- Hyper-parameter tuning is an extra cost which cannot be ignored for adopting neural network based estimators.

Chapter 5

Are Learned Methods Ready for Dynamic Environments?

Data updates in databases occur frequently, leading to a "dynamic" environment for cardinality estimators. In this chapter, we aim to answer a new question: *Are learned methods ready for dynamic environments?* We first discuss how learned methods perform against DBMSs in dynamic environments, then explore the trade-off between the number of updating epochs and accuracy, and finally investigate how much GPU can help learned methods.

5.1 Setup

Dynamic Environment. In a dynamic environment, both model accuracy and updating time matter. Consider a time range $[0, T]$. Suppose that there are n queries uniformly distributed in this time range. Suppose that given a trained initial model, the model update starts at timestamp 0 and finishes at timestamp t_u ($t_u \leq T$). For the first $n \cdot \frac{t_u}{T}$ queries, their cardinalities will be estimated using the stale model. For the remaining $n \cdot (1 - \frac{t_u}{T})$ queries, the updated model will be used.

Figure 5.1 shows an example. Suppose $T = 100$ mins and Naru spends $t_u = 75$ mins updating its model. Then, Naru needs to estimate the cardinalities for 75% (25%) of the queries using the stale (updated) model. Since many queries will be handled by the (inaccurate) stale model, although Naru performs the best in the static environment, this may not be the case in this dynamic environment.

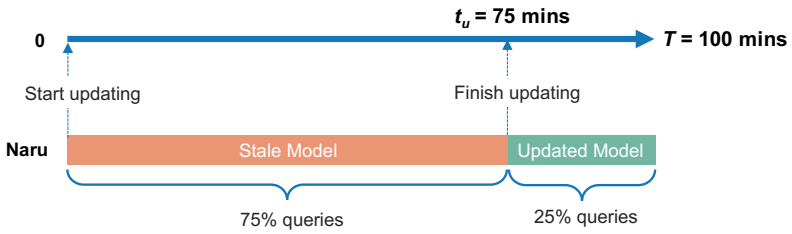


Figure 5.1: An illustration of a dynamic environment.

Dataset & Workload & Metric We use the same four real-world datasets as Chapter 4. We append 20% new data to the original dataset and apply our workload generation method to the updated data to general 10K test queries. That is, there are 10K queries uniformly distributed in $[0, T]$. Here, T is a parameter in our dynamic environment. Intuitively, it represents how "frequent" the data is being updated. For example, if the data is periodically updated every 100 mins, then we can set $T = 100$ mins. We report the 99th percentile q-error of the 10K queries.

Data Update. We ensure that the appended 20% new data has different correlation characteristics from the original dataset. Otherwise, the stale model may still perform well and there is no need to update the model. To achieve this, we create a copy of the original dataset and sort each column individually in ascending order, which leads to the maximum Spearman's rank correlation between every pair of columns. We randomly pick up 20% of the tuples from this copied dataset and append them to the original dataset.

Model Update. The initial models we use are the same as Chapter 4, which are tuned towards a better accuracy. We follow the original papers of the learned methods to update their models. Naru and DeepDB are trained on data. As described in their papers, Naru is updated by one epoch, while DeepDB is updated by inserting a small sample (1%) of the appended data to its tree model. MSCN and LW-XGB/NN use query results as training data. Since the updating procedure is not discussed in the original MSCN paper, we adopt LW-XGB/NN's updating procedure for MSCN. After generating a training workload, we use a sample (5% of the original datasets) to update the query label. LW-XGB and LW-NN originally use 8K and 16K queries for updating accordingly. We assign 10K queries for MSCN as a fair size of training data.

Note that the updating time is different from the training time presented in Figure 4.2. To update a model quickly, the updating time involves fewer epochs. Also, for query driven methods, they need to add the query results' updating time because this is a major difference between data-driven and query-driven learned methods.

5.2 Which Method Performs the Best in Dynamic Environments?

In this experiment, we test 5 learned methods against 3 DBMSs on CPU. We vary T for each dataset to represent different update frequencies: high, medium, low. Note that our four datasets are different in size, so T is set differently for each dataset. The results are shown in Figure 5.2. If a model cannot finish within T, we will put "$\times$" in the figure.

We first compare DBMSs with learned methods. We can see that DBMSs have more stable performance than learned methods by varying T. The reason is that DBMSs have very short updating time and almost all the queries are run on their updated statistics.

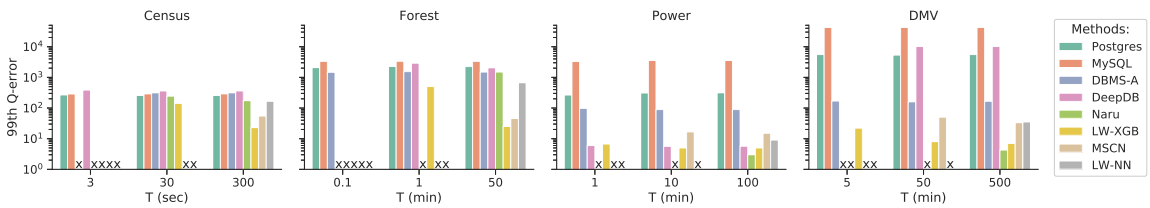


Figure 5.2: Comparison of learned methods and DBMSs under different dynamic environments on four datasets.

We also observe that many learned methods cannot catch up with fast data updates. Even if they can, they do not always outperform DBMSs. For example, when $T = 50$ mins on DMV, DBMS-A outperforms DeepDB by about 100× since the updated DeepDB model cannot capture correlation change well.

We then compare different learned methods. Overall, LW-XGB can perform better or at least comparable with others in most cases. MSCN and LW-NN do not perform well since they need longer updating time and the stale models process too many queries. Recall that Naru has a very good accuracy when there is no update. In dynamic environments, however, Naru does not outperform LW-XGB when update frequencies are high or medium. Naru has a similar performance with DBMSs on Census and Forest. This is because Naru uses 1 epoch to update its model. Although it enables a shorter updating time, 1 epoch is not enough to have good accuracy for Census and Forest datasets. For DMV, we have the same observation as [18]. Naru performs well on DMV within 1 epoch. We will discuss this trade-off between updating epochs and accuracy in the next subsection. DeepDB usually has a very short updating time. However, its updated model cannot capture the correlation change well, thus it does not outperform LW-XGB/NN in most cases.

In terms of updating time, there is no all-time winner on different datasets. For example, on Census, DeepDB (data driven) is the fastest method, whereas on DMV, LW-XGB (query driven) is the fastest one, although these two methods are the top-2 fastest methods in this experiment. The reason behind this is that the updating time of data driven methods is usually proportional to the size of the data. Intuitively, data driven methods compress the information of the data to the models to represent the joint distribution. When the size of the data gets larger, the complexity of the model should be higher and harder to train. In contrast, query driven methods have the training overhead of generating query labels. However, given a larger dataset and a fixed number of training queries, the complexity of their models do not necessarily become higher. In practice, the choice of using data or query driven methods is really subjective to the applications.

5.3 What Is the Trade-off Between Updating Time and Accuracy?

We explore the trade-off between the number of updating epochs and accuracy for learned methods. Due to the space limit, we only show Naru's results on Census and Forest to illustrate this point.

We set $T = 10$ mins on Census and $T = 100$ mins on Forest to ensure Naru with different epochs can finish updating within T. Figure 5.3 shows our results. "Stale" represents the stale model's performance on 10K queries. "Updated" represents the updated model's performance. "Dynamic" represents the Naru's performance (the stale model first and then the updated model) on 10K queries. We can see a clear trade-off of Naru on Forest. That is, "Dynamic" first goes down and then goes up. The reason is that long training time (epochs) makes the model update slow. It leaves more queries executed using the stale mode. Even though more epochs improve the updated model's performance, it hurts the overall performance.

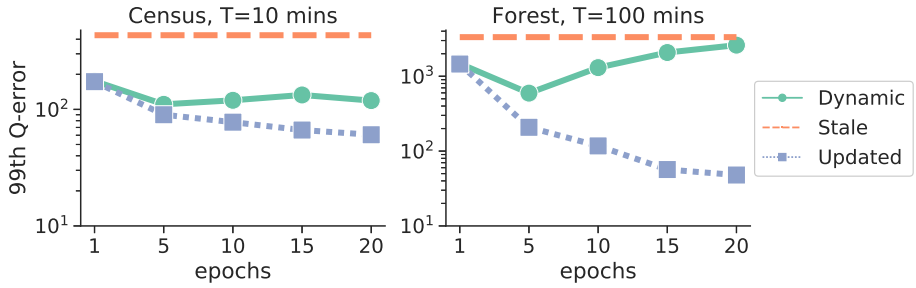


Figure 5.3: Trade-off (Naru): epochs vs accuracy.

In this Naru experiment, we show the trade-off between updating time and accuracy by varying the number of epochs. There are other ways to achieve this trade-off. For example, for query-driven methods, they need to update the answers to a collection of queries. Using sampling is a nice way to reduce the updating, but it will lead to approximate answers, thus hurting the accuracy. It is an interesting research direction to study how to balance the trade-off for learned methods.

5.4 How Much Does GPU Help?

We explore how much GPU can help Naru and LW-NN. We set $T = 100$ mins on Forest and $T = 500$ mins on DMV to ensure they can finish updating within T. The results are shown in Figure 5.4.

We can see that with the help of GPU, LW-NN is improved by around 10× and 2× on Forest and DMV, respectively. There are two reasons for these improvements: (1) LW-NN's training time can be improved by up to 20× with GPU; (2) A well-trained LW-NN (500

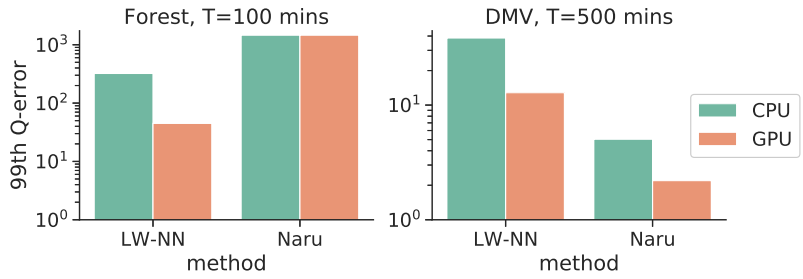


Figure 5.4: GPU affects the performance.

epochs) has a good accuracy. For Naru, it is improved by 2× on DMV. However, it does not get improved on Forest. This is because that 1 epoch is not enough for Naru to get a good updated model on Forest, although shorter updating time leaves more queries for the updated model.

5.5 Main Findings

Our main findings of this chapter are summarized as follows:

- Learned methods cannot catch up with fast date updates. MSCN, LW-NN, Naru, and DeepDB return large error in dynamic environments for different reasons.
- Within learned methods, there is no clear winner. Naru performs the best when date updates are not frequent, while LW-XGB performs the best in more dynamic environments.
- In terms of updating time, DeepDB is the fastest data-driven method and LW-XGB is the fastest query-driven method, but there is no clear winner between DeepDB and LW-XGB.
- There is a trade-off between updating time and accuracy for learned methods. It is not easy to balance the trade-off in practice and requires more research efforts on this topic.
- GPU is able to, but not necessarily, improve the performance. It is important to design a good strategy to handle model updates in order to benefit from GPU.

Chapter 6

When Do Learned Estimators Go Wrong?

One advantage of simple traditional methods like histogram and sampling is their transparency. We know that when the assumptions (e.g., attribute-value-independence (AVI), uniform spread) made by these estimators are violated, they tend to produce large q-errors. In comparison, learned estimators are opaque and lack understanding. In this chapter, we seek to explore scenarios when learned methods do not work well. We run a micro-benchmark to observe how their large error changes when we alter the underlying dataset. We also identify some logical rules that are simple and intuitive but are frequently violated by these learning models.

6.1 Setup

Dataset. We introduce our synthetic dataset generation procedure. We generate datasets with two columns by varying three key factors: *distribution* (of the first column), *correlation* (between the two columns) and *domain size* (of the two columns). Each dataset contains 1 million rows.

The first column is generated from the genparato function in scikit-learn [66], which can generate random numbers from evenly distributed to very skewed. We vary the distribution parameter s from 0 to 2, where $s = 0$ represents uniform distribution and the data becomes more skewed as s increases.

The second column is generated based on the first column in order to control the correlation between the two columns. We use $c \in [0, 1]$ to represent how correlated the two columns are. For each row (v_1, v_2), we set v_2 to v_1 with a probability of c and set v_2 to a random value drawn from the domain of the first column with a probability of $1 - c$. Obviously, the two columns are independent when $c = 0$. They are more correlated as c increases and become functional dependent when $c = 1$.

We also consider domain size d (the number of distinct values), which is related to the amount of information contained in a dataset. It can affect the size needed to encode the space for models like Naru. To control the domain size, we convert the generated continuous values into bins. In our experiment, we generate datasets with domain size 10, 100, 1K and 10K.

Workload. Since the goal of this experiment is to study the cases when learned methods go wrong, we generate center values from each column's domain independently (OOD) for all the queries in order to explore the whole query space and find as many hard queries as possible. Other workload generation settings are the same as Chapter 4.

Hyper-parameter Tuning. We adopt the default hyper-parameters recommended in [32] (RDC threshold = 0.3 and minimum instance slice = 0.01) for DeepDB and fix the tree size of LW-XGB to 128. As for neural network models, we randomly pick up three hyper-parameter settings with 1% size budget using the same way as Chapter 4 and select one that consistently reports good results.

6.2 When Do Learned Estimators Produce Large Error?

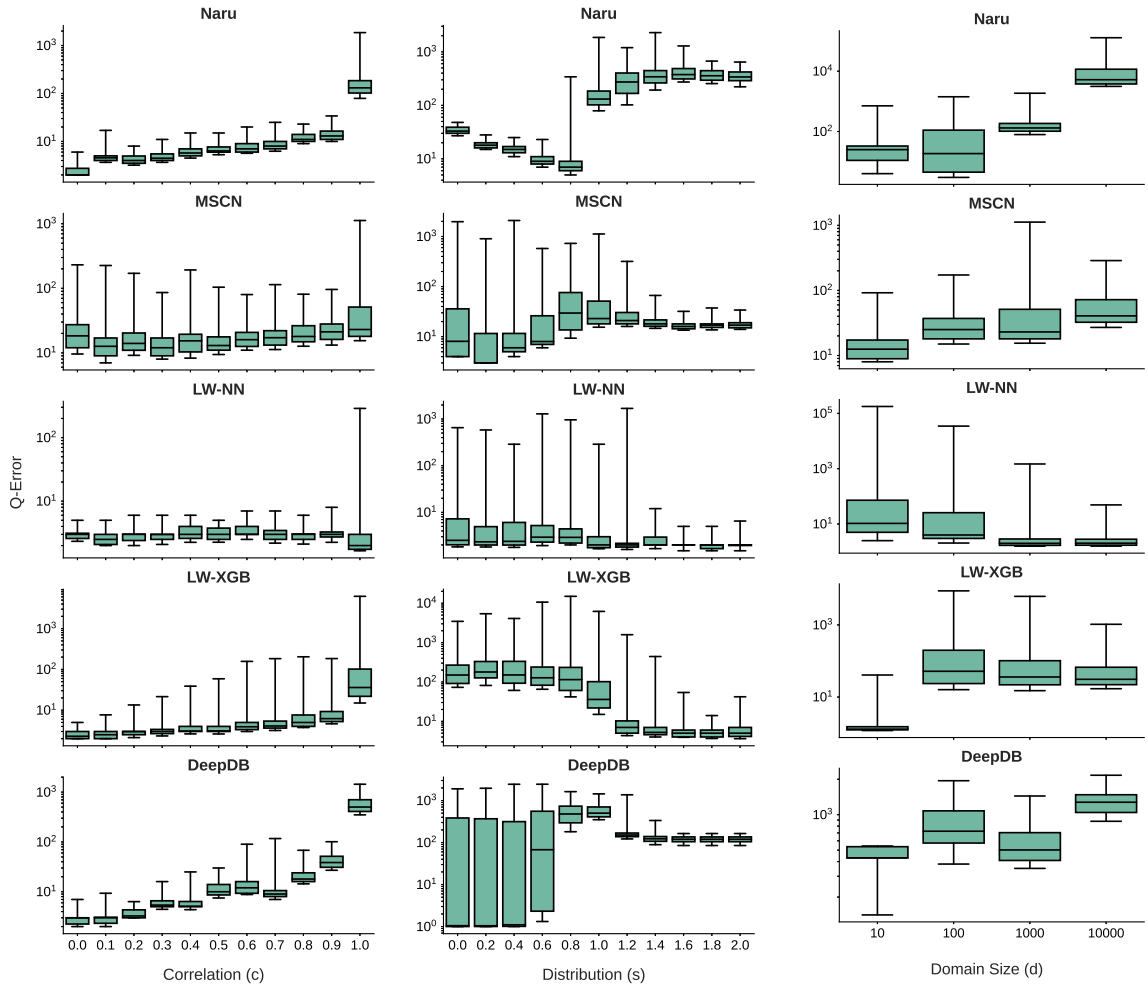
We examine how the accuracy of learned models will be affected by different factors. We train the exact same model on datasets with only one factor varied and the other two fixed, and use the same 10K queries to test the models. Instead of comparing different models, here we aim to observe the performance change for the same model on different datasets. We only exhibit the distribution of the top 1% q-errors to make the trend on large errors more clear.

Correlation. A common thing we found when we vary the correlation parameter c is that all methods tend to produce larger q-error on more correlated data. Figure 6.1a shows the top 1% q-error distribution trend on different correlation degrees with the first column distribution $s = 1.0$ (exponential distribution) and domain size $d = 1000$. It is clear that boxplots in all the figures have a trend to go up when c increases.

Another observation is that the q-error of all estimators rises dramatically ($10 \sim 100\times$) when two columns become functional dependent ($c = 1.0$). This pattern commonly exists on different pairs of s and d values we tested, which indicates that there is space to improve theses learned estimators on highly correlated datasets especially when functional dependency exists.

Distribution. Each learned method reacts differently when we change the distribution of the first column. Figure 6.1b shows the top 1% q-error distribution trend when s goes from 0.0 to 2.0 while fixing the correlation $c = 1.0$ and domain size $d = 1000$.

In general, Naru outputs larger max q-errors when data is more skewed ($s > 1.0$), while MSCN, LW-XGB/NN and DeepDB show an opposite pattern. We suspect this difference



(a) $s = 1.0$, $d = 1000$ (b) $c = 1.0$, $d = 1000$ (c) $s = 1.0$, $c = 1.0$

Figure 6.1: Top 1% q-error distribution under different correlations (a), distributions (b) and domain size (c).

might be caused by the different basic building blocks used in each method. The common thing shared within the latter approaches is that they all incorporate basic synopsis like sampling or 1D histogram in their models. These statistics might directly record a relatively accurate cardinality for the query involving a frequent value in the dataset, and thus reduce the max error when data is very skewed. If this is true, we can study how to incorporate a similar idea into Naru and make it more robust on skewed data.

Another interesting thing is that unlike max q-error, the 99th percentile q-error (the lower extreme of the boxplot since we only report top 1% q-errors) shows an opposite pattern on MSCN and DeepDB. Here we guess that for both methods, it might be because of the number of queries with very small selectivity increases when s increases. In such cases, the sample feature in MSCN would remain in all zero on many queries, which is not very useful. As for DeepDB, since its leaf node has the AVI assumption, it would produce very large result when the selectivity of each predicate is large but the combined result is very small, which is common when s is large.

Domain Size. Figure 6.1c shows the top 1% q-error distribution on datasets generated under different domain size ($s = 1.0$ and $c = 1.0$). Notice that Naru may use a different model architecture on each domain size to meet the same 1% size budget.

Except for LW-NN, all methods output larger error on larger domain size. Naru exhibits a 100× performance degrade when domain size goes from 1K to 10K. This may be because that the embedding matrix on 10K domain occupies a big portion of the size budget and thus the rest of the model does not have enough capacity to learn the data distribution. Having a more efficient encoding method could mitigate this issue for Naru. LW-XGB shows a very strong result when domain size is 10 and the error becomes 100× bigger on larger domains. MSCN and DeepDB are relatively more robust than other methods but still experience around 10× degrade when domain size increases from 10 to 10K.

It is interesting to see that LW-NN and LW-XGB show opposite trend even though they share the same input feature and optimization goal. It is very likely that this phenomenon is caused by the underlying model they adopt. We suspect that the input query space becomes more "discrete" when the domain size is as small as 10. Therefore a small change in the query predicate can dramatically change the cardinality result or might not affect it at all. It can be hard for the neural network used in LW-NN to learn since compared with the tree-based model in LW-XGB, neural network intuitively fits the data in a more smooth and continuous way.

6.3 Do Learned Estimators Behave Predictably?

During our experimental study, we identify some *illogical behaviors* from some of the learned models. For example, when we changed one of the query predicates from $[320, 800]$ to a

smaller range $[340, 740]$, the real cardinality decreased, but the estimated cardinality by LW-XGB unexpectedly increased by 60.8%.

This kind of unreasonable behavior caught our attention. The violation of simple logical rules like this could cause troubles for both DBMS developers and users (see Section 6.4 for more discussion). Inspired by the work [82] in the deep learning explanation field, we propose five basic rules for cardinality estimation. These rules are simple and intuitive which the users may expect cardinality estimators to satisfy:

1. **Monotonicity:** With a stricter (or looser) predicate, the estimation result should not increase (or decrease).

2. **Consistency:** The prediction of a query should be equal to the sum of the predictions of queries split from it (e.g. a query with predicate $[100, 500]$ on A_i can be split to two queries with $[100, 200)$ and $[200, 500]$ on A_i respectively and other predicates remain the same).

3. **Stability:** For any query, the prediction result from the same model should always be the same.

4. **Fidelity-A:** The selectivity estimation should be equal to 1 for querying on the entire domain (e.g. SELECT * FROM R WHERE $min_i \leq A_i \leq max_i$).

5. **Fidelity-B:** The estimation should be 0 for a query with an invalid predicate (e.g. SELECT * FROM R WHERE $100 \leq A_i \leq 10$).

According to these proposed rules, we check each learned estimator and summarize whether it satisfies or violates each rule in Table 6.1. Some of the rules like Fidelity-B can be fixed with some simple checking mechanisms, however here we only consider the original output of the underlying model used in each estimator in order to see whether these models behave in a logical way natively.

Table 6.1: Satisfaction and violation of rules by learned estimators. (✓: satisfied, ×: violated)

Rule	Naru	MSCN	LW-XGB	LW-NN	DeepDB
Monotonicity	×	×	×	×	✓
Consistency	×	×	×	×	✓
Stability	×	✓	✓	✓	✓
Fidelity-A	✓	×	×	×	✓
Fidelity-B	✓	×	×	×	✓

Naru's progressive sampling technique introduces uncertainty to the inference process, which causes the violation of stability. Specifically, we find that the estimations on the same query can vary dramatically when two columns are functional dependent and the query predicate covers a large range on the first column while only a few values on the second

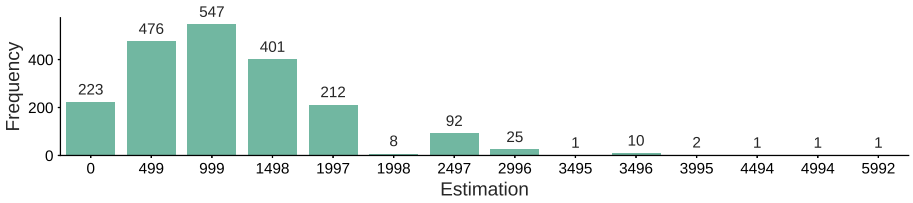


Figure 6.2: Prediction result of running Naru on the same query 2000 times ($s = 0.0$, $c = 1.0$, $d = 1000$).

column. It is because the variance of the conditional probabilities that Naru would sample during inference is very large. Figure 6.2 shows an example of the estimation results using Naru to run a query (the actual cardinality is 1036) for 2000 times under this setting. The results are spread over the range of [0, 5992]. This instability also causes Naru to violate monotonicity and consistency rules.

The regression-based methods (MSCN, LW-NN, LW-XGB) violate all the rules except for stability. It is not a very surprising result since there is no constraint enforced to the model during both training and inference stages. In comparison, DeepDB does not violate any rules since it is built on basic histograms and the computation between nodes is restricted to addition and multiplication.

6.4 What Will Go Wrong in Production?

We discuss four issues that may appear when deploying (black-box and illogical) learned models in production.

Debuggability. It is challenging to debug black-box models like Naru, MSCN and LW-XGB/NN. Firstly, black-box models may fail silently, thus there is a high risk to miss a bug. For example, if there is a bug in the hyper-parameter tuning stage, the model can still be trained and may pass all test cases. Secondly, black-box models make it hard to trace an exception back to the actual bug. If the learned model produces a large error for a given query, it is difficult to tell whether it is a normal bad case or caused by a bug in the code or training data.

Explainability. Another related issue is that black-box models lack explainability. It brings some challenges for query optimizer version update. We might find a model architecture or hyper-parameter method improve the estimation accuracy and want to add it to the new version. However, it is hard to explain to the database users about which type of query and what kind of scenario will be affected by this upgrade.

Predicability. Since learned methods do not follow some basic logic rules, the database system may behave illogically, thus confusing database users. For example, a user would expect a query to run faster by adding more filter conditions. Due to the violation of the

monotonicity rule, this may not be the case when the database system adopts a learned model like Naru, MSCN, or LW-XGB/NN.

Reproducibility. It is common that a database developer wants to reproduce customers' issues. In order to reproduce the issues, the developer needs information, such as the input query, optimizer configurations, and metadata [79]. However, if the system adopts Naru which violates the stability rule, it would be hard to reproduce the result due to the stochastic inference process.

6.5 Main Findings

Our main findings of this chapter are summarized as follows:

- All new learned estimators tend to output larger error on more correlated data, and the max q-error jumps quite dramatically when two columns are functional dependent.
- Different methods react differently for more skewed data or for data with larger domain size. This might be due to the differences in the choice of models, input features, and loss functions.
- We propose five rules for cardinality estimators and find that all new learned models except for DeepDB violate these rules.
- The non-transparency of the models used in new learned estimators can be troublesome in terms of debuggability, explainability, predicabiltiy, and reproducibility when deployed in production.

Chapter 7

Conclusion

In this thesis, we raised an important but unexplored question: "Are we ready for learned cardinality estimation?". We surveyed seven new learned methods and put them into a taxonomy. We found that existing experimental studies are inadequate to answer this question. In response, we proposed a unified workload generator and conducted comprehensive experiments on four real-world and one synthetic datasets. We explored whether learned methods are ready for both static environments and dynamic environments, and dived into when learned methods may go wrong.

We concluded that new learned methods are more accurate than traditional methods. However, in order to put them in a well-developed system, there are many missing parts to be resolved, such as low speed in training and inference, hyper-parameter tuning, black-box property, illogical behaviors, and dealing with frequent data updates. As a result, the current learned methods are still not ready to be deployed in a real DBMS. Overall, this is an important and promising direction to be further explored by the database community.

Chapter 8

Future Work

We have showed that the high cost (Chapter 4 and Chapter 5) and the non-transparency (Chapter 6) are the two main challenges of applying learned carnality estimators in DBMS. What can we do in order to close these gaps? In this chapter, we discuss future work opportunities for learned cardinality estimation in the two research directions. We also further discuss the limitation of this experimental study and what can be done in the next stage.

8.1 Research Opportunity

We propose the two research directions targeting on the two disadvantages of learned methods: high cost and non-transparency.

8.1.1 Control the Cost of Learned Estimators

Balance the Efficiency-Accuracy Tradeoff. Balancing the tradeoff between accuracy and training (updating) time as well as inference latency can be an interesting aspect to start with. To retrain a model, simple approximate methods like using a sample instead of full data to calculate the queries' ground-truth or incrementally updating the model, can be leveraged to make neural network models more efficient. Similar ideas in machine learning techniques such as early stop [8] and model compression [11] can also be used to reduce the cost.

Ensemble methods can also be a way to balance this tradeoff. A fast but less accurate method can be used as a temporary replacement when the slow but accurate model is not ready. Another idea is to apply multiple approaches in a hierarchical fashion. For example, if a query is less complex (e.g., having fewer predicates [75]), we can use lightweight methods to estimate the cardinality, otherwise we choose the heavy but accurate one.

Hyper-parameter Tuning for Learned Estimators. Hyper-parameter tuning is crucial for new learned models to achieve high accuracy. Algorithms like random search [5], bayesian

optimization [76], and bandit-based approaches [47] can be adopted to reduce the cost of obtaining a good hyper-parameter configuration.

Meta-learning tackles the hyper-parameter tuning problem in a "learning to learn" fashion [20, 22, 4]. The basic idea is to learn from a wide range of learning tasks and solve new similar tasks using only a few training samples. Specifically, when we want to train a model for cardinality estimation on a new dataset or a new workload, there is no need to start entirely from scratch. Instead, we can leverage our previous learning experience, such as the relationship between dataset characteristics and good hyper-parameter sets, in order to obtain a good configuration more efficiently.

Another aspect for hyper-parameter tuning is the goal of tuning. Usually, the goal is to find the configuration with the best accuracy/loss. In the cardinality estimation setting, it is worth doing more exploration to take training/updating time into consideration, because of the trade-off above.

8.1.2 Make Learned Estimators Trustworthy

Interpret Learned Estimators. There have been extensive works in machine learning explanation trying to understand why a model makes a specific prediction for a specific input, such as surrogate models [73], saliency maps [77], influence function [37], decision sets [42], rule summaries [74], and general feature attribution methods [83, 53]. These techniques could be leveraged to interpret black box cardinality estimators to some extend. For example, when we get a large error for a query during the test phase, we can use influence function [37] to find the most influential training examples, or we can use shapely value [53] to check the importance of each input feature. However, how effective these methods are in the cardinality estimation setting is still an open problem.

Handle Illogical Behaviours. Our study shows that many learned methods do not behave logically. One way to handle this is to define a complete set of logical rules and identify which rules are violated for a certain method. This will add more transparency to each learned method and enable the database developers to know what kind of behavior can be expected from each method. The logical rules we propose in Section 6.3 can be seen as an effort from this perspective. Another way is to enforce logical rules as constraints for model design. There are some existing works in the machine learning community [38, 12, 21]. Similar ideas could be applied to the design of cardinality estimation models.

8.2 Limitation and Future Work

In this section, we present some limitations of this experimental study and what we can do in the next stage:

- We only focus on the problem of single table cardinality estimation. The cardinality of join query is a more challenging problem and there are many works trying to improve existing solutions (Section 2.2). As future work, we plan to extend our experimental study to join query and identify new challenges that are caused by the join operation for learned methods.
- For the evaluation in dynamic environment (Chapter 5). We construct synthetic data update and do not consider cases when data is static but query workload changes. In the next step, we plan to explore and conduct experiments on real-world data updates. We will also include workload drift scenarios in our future work.
- For now, we only examine each method to see whether they satisfy a logic rule or not (Section 6.3). For methods that violate the rules, our next plan is to design experiments to show how frequently these rules are violated.
- We plan to do an end-to-end evaluation in our next step to see how the accuracy improvement and high cost of learned methods can actually impact the overall performance including both query optimization and execution.

Bibliography

[1] Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 181–192. ACM Press, 1999.

[2] Christos Anagnostopoulos and Peter Triantafillou. Learning to accurately COUNT with query-driven predictive analytics. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, pages 14–23. IEEE Computer Society, 2015.

[3] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 221–230. ACM, 2018.

[4] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 459–468. PMLR, 2017.

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

[6] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Stholes: A multidimensional workload-aware histogram. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 211–222. ACM, 2001.

[7] Walter Cai, Magdalena Balazinska, and Dan Suciu. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 18–35. ACM, 2019.

[8] Rich Caruana, Steve Lawrence, and C. Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In Todd K. Leen, Thomas G.

Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 402–408. MIT Press, 2000.

[9] Chung-Min Chen and Nick Roussopoulos. Adaptive selectivity estimation using query feedback. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA, May 24-27, 1994*, pages 161–172. ACM Press, 1994.

[10] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[11] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *CoRR*, abs/1710.09282, 2017.

[12] Jan Chorowski and Jacek M. Zurada. Learning understandable neural networks with nonnegative weight constraints. *IEEE Trans. Neural Networks Learn. Syst.*, 26(1):62–69, 2015.

[13] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theory*, 14(3):462–467, 1968.

[14] Amol Deshpande, Minos N. Garofalakis, and Rajeev Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 199–210. ACM, 2001.

[15] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In Armand Prieditis and Stuart J. Russell, editors, *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 194–202. Morgan Kaufmann, 1995.

[16] Dheeru Dua and Casey Graff. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2017.

[17] Anshuman Dutt, Chi Wang, Vivek R. Narasayya, and Surajit Chaudhuri. Efficiently approximating selectivity functions using low overhead regression models. *Proc. VLDB Endow.*, 13(11):2215–2228, 2020.

[18] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. Selectivity estimation for range predicates using lightweight models. *Proc. VLDB Endow.*, 12(9):1044–1057, 2019.

[19] Changyong FENG, Hongyue WANG, Naiji LU, Tian CHEN, Hua HE, Ying LU, and Xin M TU. Log-transformation and its implications for data analysis. *Shanghai Archives of Psychiatry*, 26(2):105, 2014.

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

[21] William Fleshman, Edward Raff, Jared Sylvester, Steven Forsyth, and Mark McLean. Non-negative networks against adversarial attacks. *CoRR*, abs/1806.06108, 2018.

[22] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1563–1572. PMLR, 2018.

[23] Allen Van Gelder. Multiple join size estimation by virtual domains. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC, USA*, pages 180–189. ACM Press, 1993.

[24] Rainer Gemulla. *Sampling algorithms for evolving datasets*. PhD thesis, Dresden University of Technology, Germany, 2008.

[25] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: masked autoencoder for distribution estimation. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 881–889. JMLR.org, 2015.

[26] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 461–472. ACM, 2001.

[27] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 463–474. ACM, 2000.

[28] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *VLDB J.*, 14(2):137–154, 2005.

[29] Hazar Harmouch and Felix Naumann. Cardinality estimation: An experimental survey. *Proc. VLDB Endow.*, 11(4):499–512, 2017.

[30] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. Deep learning models for selectivity estimation of multi-attribute queries.

In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1035–1050. ACM, 2020.

[31] Max Heimel, Martin Kiefer, and Volker Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1477–1492. ACM, 2015.

[32] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. Deepdb: Learn from data, not from queries! *Proc. VLDB Endow.*, 13(7):992–1005, 2020.

[33] H. V. Jagadish, Hui Jin, Beng Chin Ooi, and Kian-Lee Tan. Global optimization of histograms. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 223–234. ACM, 2001.

[34] Alekh Jindal, Shi Qiao, Hiren Patel, Zhicheng Yin, Jieming Di, Malay Bag, Marc Friedman, Yifung Lin, Konstantinos Karanasos, and Sriram Rao. Computation reuse in analytics job service at microsoft. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 191–203. ACM, 2018.

[35] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. Estimating join selectivities using bandwidth-optimized kernel density models. *Proc. VLDB Endow.*, 10(13):2085–2096, 2017.

[36] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org, 2019.

[37] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894. PMLR, 2017.

[38] Aleksander Kołcz and Choon Hui Teo. Feature weighting for improved classifier robustness. In *CEAS'09: sixth conference on email and anti-spam*, 2009.

[39] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 489–504. ACM, 2018.

[40] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. Learning to optimize join queries with deep reinforcement learning. *CoRR*, abs/1808.03196, 2018.

[41] Ani Kristo, Kapil Vaidya, Ugur Çetintemel, Sanchit Misra, and Tim Kraska. The case for a learned sorting algorithm. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1001–1016. ACM, 2020.

[42] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1675–1684. ACM, 2016.

[43] M. Seetha Lakshmi and Shaoyu Zhou. Selectivity estimation in extensible databases - A neural network approach. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 623–627. Morgan Kaufmann, 1998.

[44] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, 2015.

[45] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org, 2017.

[46] G Peter Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27(2):192–203, 1978.

[47] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017.

[48] Lipyeow Lim, Min Wang, and Jeffrey Scott Vitter. SASH: A self-adaptive histogram set for dynamically changing workloads. In Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer, editors, *Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, September 9-12, 2003*, pages 369–380. Morgan Kaufmann, 2003.

[49] Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. Practical selectivity estimation through adaptive sampling. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, May 23-25, 1990*, pages 1–11. ACM Press, 1990.

[50] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. Cardinality estimation using neural networks. In Jordan Gould, Marin Litoiu, and Hanan Lutfiyya, editors, *Proceedings of 25th Annual International Conference on Computer Science and Software Engineering, CASCON 2015, Markham, Ontario, Canada, 2-4 November, 2015*, pages 53–59. IBM / ACM, 2015.

[51] David López-Paz, Philipp Hennig, and Bernhard Schölkopf. The randomized dependence coefficient. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 1–9, 2013.

[52] Hongjun Lu and Rudy Setiono. Effective query size estimation using neural networks. *Appl. Intell.*, 16(3):173–183, 2002.

[53] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 4765–4774, 2017.

[54] Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. In Rajesh Bordawekar and Oded Shmueli, editors, *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, pages 3:1–3:4. ACM, 2018.

[55] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *Proc. VLDB Endow.*, 12(11):1705–1718, 2019.

[56] Volker Markl, Nimrod Megiddo, Marcel Kutsch, Tam Minh Tran, Peter J. Haas, and Utkarsh Srivastava. Consistently estimating the selectivity of conjuncts of predicates. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 373–384. ACM, 2005.

[57] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Wavelet-based histograms for selectivity estimation. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 448–459. ACM Press, 1998.

[58] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proc. VLDB Endow.*, 2(1):982–993, 2009.

[59] Magnus Müller, Guido Moerkotte, and Oliver Kolb. Improved selectivity estimation by combining knowledge from sampling and synopses. *Proc. VLDB Endow.*, 11(9):1016–1028, 2018.

[60] M. Muralikrishna and David J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In Haran Boral and Per-Åke Larson, editors, *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 1-3, 1988*, pages 28–36. ACM Press, 1988.

[61] State of New York. Vehicle, snowmobile, and boat registrations. `catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registration`, 2019. Accessed: 2019-03-01.

[62] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. Learning state representations for query optimization with deep reinforcement learning. In Sebastian Schelter, Stephan Seufert, and Arun Kumar, editors, *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, DEEM@SIGMOD 2018, Houston, TX, USA, June 15, 2018*, pages 4:1–4:4. ACM, 2018.

[63] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. An empirical analysis of deep learning for cardinality estimation. *CoRR*, abs/1905.06425, 2019.

[64] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1017–1033. ACM, 2020.

[65] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[67] Wendel Góes Pedrozo, Júlio César Nievola, and Deborah Carvalho Ribeiro. An adaptive approach for index tuning with learning classifier systems on hybrid storage environments. In Francisco Javier de Cos Juez, José Ramón Villar, Enrique A. de la Cal, Álvaro Herrero, Héctor Quintián, José Antonio Sáez, and Emilio Corchado, editors, *Hybrid Artificial Intelligent Systems - 13th International Conference, HAIS 2018, Oviedo, Spain, June 20-22, 2018, Proceedings*, volume 10870 of *Lecture Notes in Computer Science*, pages 716–729. Springer, 2018.

[68] Matthew Perron, Zeyuan Shang, Tim Kraska, and Michael Stonebraker. How I learned to stop worrying and love re-optimization. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pages 1758–1761. IEEE, 2019.

[69] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984*, pages 256–276. ACM Press, 1984.

[70] Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. In *IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011*, pages 689–690. IEEE Computer Society, 2011.

[71] Viswanath Poosala and Yannis E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 486–495. Morgan Kaufmann, 1997.

[72] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 294–305. ACM Press, 1996.

[73] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. ACM, 2016.

[74] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1527–1535. AAAI Press, 2018.

[75] Matteo Riondato, Mert Akdere, Ugur Çetintemel, Stanley B. Zdonik, and Eli Upfal. The vc-dimension of SQL queries and selectivity estimation through sampling. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part II*, volume 6912 of *Lecture Notes in Computer Science*, pages 661–676. Springer, 2011.

[76] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE*, 104(1):148–175, 2016.

[77] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML*

2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 2017.

[78] Mohamed A. Soliman, Lyublena Antova, Venkatesh Raghavan, Amr El-Helw, Zhongxian Gu, Entong Shen, George C. Caragea, Carlos Garcia-Alvarado, Foyzur Rahman, Michalis Petropoulos, Florian Waas, Sivaramakrishnan Narayanan, Konstantinos Krikellas, and Rhonda Baldwin. Orca: a modular query optimizer architecture for big data. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 337–348. ACM, 2014.

[79] Mohamed A. Soliman, Lyublena Antova, Venkatesh Raghavan, Amr El-Helw, Zhongxian Gu, Entong Shen, George C. Caragea, Carlos Garcia-Alvarado, Foyzur Rahman, Michalis Petropoulos, Florian Waas, Sivaramakrishnan Narayanan, Konstantinos Krikellas, and Rhonda Baldwin. Orca: a modular query optimizer architecture for big data. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 337–348. ACM, 2014.

[80] Utkarsh Srivastava, Peter J. Haas, Volker Markl, Marcel Kutsch, and Tam Minh Tran. ISOMER: consistent histogram construction using query feedback. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 39. IEEE Computer Society, 2006.

[81] Ji Sun and Guoliang Li. An end-to-end learning-based cost estimator. *Proc. VLDB Endow.*, 13(3):307–319, 2019.

[82] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 2017.

[83] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 2017.

[84] Jian Tan, Tieying Zhang, Feifei Li, Jie Chen, Qixing Zheng, Ping Zhang, Honglin Qiao, Yue Shi, Wei Cao, and Rui Zhang. ibtune: Individualized buffer tuning for large-scale cloud databases. *Proc. VLDB Endow.*, 12(10):1221–1234, 2019.

[85] Hien To, Kuorong Chiang, and Cyrus Shahabi. Entropy-based histograms for selectivity estimation. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 1939–1948. ACM, 2013.

[86] Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1153–1170. ACM, 2019.

[87] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. Lightweight graphical models for selectivity estimation without independence assumptions. *Proc. VLDB Endow.*, 4(11):852–863, 2011.

[88] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[89] Lucas Woltmann, Claudio Hartmann, Maik Thiele, Dirk Habich, and Wolfgang Lehner. Cardinality estimation with local deep learning models. In Rajesh Bordawekar and Oded Shmueli, editors, *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*, pages 5:1–5:8. ACM, 2019.

[90] Wentao Wu, Jeffrey F. Naughton, and Harneet Singh. Sampling-based query re-optimization. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1721–1736. ACM, 2016.

[91] Yi-Leh Wu, Divyakant Agrawal, and Amr El Abbadi. Using the golden rule of sampling for query estimation. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 449–460. ACM, 2001.

[92] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. NeuroCard: One cardinality estimator for all tables. volume 14, pages 61–73. VLDB Endowment, 2021.

[93] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Peter Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. Deep unsupervised cardinality estimation. *Proc. VLDB Endow.*, 13(3):279–292, 2019.

[94] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. Reinforcement learning with tree-lstm for join order selection. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1297–1308. IEEE, 2020.

[95] Mohamed Zaït, Sunil Chakkappen, Suratna Budalakoti, Satyanarayana R. Valluri, Ramarajan Krishnamachari, and Alan Wood. Adaptive statistics in oracle 12c. *Proc. VLDB Endow.*, 10(12):1813–1824, 2017.

[96] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 415–432. ACM, 2019.

[97] Qingqing Zhou. An experimental relational optimizer and executor. `https://github.com/zhouqingqing/qpmodel`. [Online; accessed 30-November-2020].

[98] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.