Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation [Experiment, Analysis & Benchmark Paper]

Yuxing Han[1,#], Ziniu Wu[1,#], Peizhi Wu[2], Rong Zhu[1,*], Jingyi Yang[2], Liang Wei Tan[2], Kai Zeng[1], Gao Cong[2], Yanzhao Qin[1,3], Andreas Pfadler[1], Zhengping Qian[1], Jingren Zhou[1], Jiangneng Li[1], Bin Cui[3]
[1]Alibaba Group, [2]Nanyang Technological University, [3]Peking University
[1]red.zr@alibaba-inc.com, [2]gaocong@ntu.edu.sg

ABSTRACT

Cardinality estimation (CardEst) plays a significant role in generating high-quality query plans for a query optimizer in DBMS. In the last decade, an increasing number of advanced CardEst methods (especially ML-based) have been proposed with outstanding estimation accuracy and inference latency. However, there exists no study that systematically evaluates the quality of these methods and answer the fundamental problem: *to what extent can these methods improve the performance of query optimizer in real-world settings, which is the ultimate goal of a CardEst method.*

In this paper, we comprehensively and systematically compare the effectiveness of CardEst methods in a real DBMS. We establish a new benchmark for CardEst, which contains a new complex real-world dataset STATS and a diverse query workload STATS-CEB. We integrate multiple most representative CardEst methods into an open-source database system PostgreSQL, and comprehensively evaluate their true effectiveness in improving query plan quality, and other important aspects affecting their applicability, ranging from inference latency, model size, and training time, to update efficiency and accuracy. We obtain a number of key findings for the CardEst methods, under different data and query settings. Furthermore, we find that the widely used estimation accuracy metric (Q-Error) cannot distinguish the importance of different sub-plan queries during query optimization and thus cannot truly reflect the generated query plan quality. Therefore, we propose a new metric P-Error to evaluate the performance of CardEst methods, which overcomes the limitation of Q-Error and is able to reflect the overall end-to-end performance of CardEst methods. It could serve as a better optimization objective for future CardEst methods. We have made all of the benchmark data and evaluation code publicly available at `https://github.com/Nathaniel-Han/End-to-End-CardEst-Benchmark`.

1 INTRODUCTION

The query optimizer is an integral component in modern DBMS. It is responsible for generating high-quality execution plans for the input SQL queries. *Cardinality estimation(CardEst)* plays a significant role in query optimization. It aims at estimating the result size of all sub-plans of each query and guiding the optimizer to select the optimal join operations. The performance of CardEst has a critical impact on the quality of the generated query plans.

Background: Due to its important role in DBMS, CardEst has been extensively studied, by both academic and industrial communities. Current open-source and commercial DBMSes mainly use two traditional CardEst methods, namely histogram, and sampling. The histogram method [42], used in PostgreSQL[9] and SQL Server[29], assumes all attributes and joins are mutually independent and builds a (cumulative) histogram on each attribute. It is simple and fast but may incur huge estimation errors as all correlations are ignored. Some variants such as multi-dimensional histograms [1, 15, 32, 51] and dependency-based histograms [8] can improve the estimation accuracy but in return incur large storage overhead. The sampling method [19, 24, 27], used in MySQL [41] and MariaDB [43], fetches a small piece of data on-the-fly for estimation. It is free of storage but has poor performance on high-dimensional data.

The core task of CardEst is to build a compact model capturing data and/or query information. With the prosperity of machine learning (ML), we witness a proliferation of learned methods for CardEst in the last three years [10, 18, 21, 25, 47, 55, 56, 59, 60, 64]. These methods could be categorized into two classes, namely query-driven and data-driven. Query-driven CardEst methods build discriminative models mapping featurized queries to their cardinalities. The most representative ones include MSCN [25] and LW-XGB/LW-NN [10], for which models are built on top of multi-set convolutional networks, gradient boosted trees, and neural networks. Data-driven CardEst methods directly model the joint distribution of all attributes. The most representative ones include deep auto-regression [59, 61], Bayesian networks [14, 50, 56], sum-product network (SPN) [21] and its variant FSPN [64]. In comparison with the traditional methods, their estimation accuracy stands out as their models are more sophisticated and fine-grained [20, 56, 61, 64].

Motivation: Despite the recent advance of the CardEst methods, we notice that a fundamental problem has not yet been answered, which is *"to what extent can these advanced CardEst methods improve the performance of query optimizers in real-world settings?"* Although existing studies have conducted extensive experiments, they suffer from the following shortcomings:

1. *The data and query workloads used for evaluation may not well represent the real-world scenarios.* The widely adopted JOB-LIGHT query workload on IMDB benchmark data [26] touches at most 8 numerical or categorical attributes within six tables, whose schema forms a star-join. The recent benchmark work [52] only evaluate these methods in a single table scenario. Therefore, the existing works are not sufficient to reflect the behavior of CardEst methods on complex real-world data with high skewness and correlations and multi-table queries with various join forms and conditions.

The first two authors contribute equally to this paper.
∗ Corresponding author.

2. *Most of the evaluations do not show the end-to-end improvement of a CardEst method on the query optimizer.* Existing works usually evaluate CardEst methods and report the results on the algorithm-level metrics, such as estimation accuracy and inference latency. These metrics only evaluate the quality of the CardEst algorithm itself, but cannot reflect how these methods behave in a real DBMS. During the query optimization process, the estimation accuracy of different sub-plan queries matters differently to the quality of the query plan [2, 39, 49]. For instance, from our benchmark results, we find that some methods with a much better estimation accuracy on average produce a much worse query plan because they mistake a few very important estimations. Some methods, although their inference latency seem to be acceptable (e.g. 100*ms* per estimation), are not practical to optimize complex queries that have tens or hundreds of sub-plan queries to estimate for one query in order to generate its query plan. Therefore, the "gold standard" to examine a CardEst method is to integrate it into the query optimizer of a real DBMS and record the *end-to-end query time*, including both query plan generation time and execution time. Unfortunately, this end-to-end evaluation has been ignored in most existing works.

To address these two problems, the DBMS community needs 1) new benchmark datasets and query workloads that can represent the real-world settings and 2) an in-depth end-to-end evaluation to analyze performance of CardEst methods.

Contributions and Findings: In this paper, we provide a systematic evaluation on representative CardEst methods, which overcomes the limitations of existing works. The main contributions of our work are listed as follows:

1. *We establish a new benchmark for CardEst that can represent real-world settings.* Our benchmark includes a real-world dataset STATS and a hand-picked query workload STATS-CEB. STATS has complex properties such as large attribute numbers, strong distribution skewness, high attribute correlations, and complicated join schema, which poses challenges to the CardEst methods. STATS-CEB contains a number of diverse multi-table join queries varying in the number of involved tables, true cardinality, and different join types (e.g., chain/star, one-to-many/many-to-many, etc.). This benchmark helps better reveal the advantages and expose the drawbacks of existing CardEst methods in real-world settings.

2. We provide an end-to-end evaluation platform for CardEst and present a comprehensive evaluation and analysis on the representative CardEst methods. We provide an approach that could integrate any CardEst method in the built-in query optimizer of PostgreSQL, a well-known open-source DBMS. Based on this, we evaluate the performance of both traditional and ML-based CardEst methods in terms of the end-to-end query time and other important aspects affecting their applicability, including inference latency, model size, training time, update efficiency, and update accuracy. From the results, we make dozen key observations (O) that reflect the performance of each CardEst method in different data and query settings. Some key findings are listed as follows:

K1. Improvement (O1): On numerical and categorical query workloads, the ML-based data-driven CardEst methods can achieve near-optimal performance, whereas the other methods can hardly improve the PostgreSQL baseline.

K2. Method (O3, O8-10): Among the data-driven methods, probabilistic graphical models have outstanding performance in both end-to-end query time and other practicality aspects because they make the right balance in tuning the strictness of independence assumption to keep inference efficiency and model effectiveness.

K3. Accuracy (O5-6, O11-13): Accurate estimation of queries with large cardinality is more important than the small ones. The accuracy metric Q-Error [31] that is used in most recent works on CardEst, cannot reflect a method's end-to-end query performance.

K4. Latency (O7): The inference latency of CardEst methods has a non-trivial impact on the end-to-end query time.

3. We propose a new metric that can indicate the overall quality of CardEst methods. Previous CardEst quality metrics, such as Q-Error, can only reflect the estimation accuracy of each (sub-plan) query. However, as different sub-plan queries do not have the same importance to the final query plan, these metrics may not accurately reflect the overall end-to-end performance of CardEst methods. This suggests that Q-Error metric may not be an effective optimization objective. Therefore, inspired by the recent work [34, 35], we propose a new metric called P-Error, which directly relates the estimation accuracy of (sub-plan) queries to the ultimate query execution plan quality of the CardEst methods. Based on our analysis, P-Error is highly correlated with the end-to-end query time improvement. Thus, it could serve as a potential substitute for Q-Error and a better optimization objective for learned CardEst methods.

Organization: The rest of paper is organized as follows. Section 2 reviews the background and preliminary of CardEst. Section 3 introduces our new benchmark and provides a detailed comparison with the existing one. Section 4 gives the settings of our evaluation plan. Sections 5–7 present the evaluation results and comprehensive analysis. Section 8 summarizes key findings and conclusions.

2 PRELIMINARIES AND BACKGROUND

In this section, we introduce some preliminaries and background, including a formal definition of the cardinality estimation (CardEst) problem, a brief review on representative CardEst algorithms and a short analysis on existing CardEst benchmarks.

CardEst Problem: In the literature, CardEst is usually defined as a statistical problem. Let T be a table with k attributes $A = \{A_1, A_2, \ldots, A_k\}$. T could either represent a single relational table or a joined table. In this paper, we assume each attribute A_i for each $1 \leq i \leq k$ to be either categorical (whose values can be mapped to integers) or continuous, whose domain (all unique values) is denoted as D_i. Thereafter, any selection query Q on T can be represented in a canonical form: $Q = \{A_1 \in R_1 \wedge A_2 \in R_2 \wedge \cdots \wedge A_n \in R_n\}$, where $R_i \subseteq D_i$ is the constraint region specified by Q over attribute A_i (i.e. filter predicates). Without loss of generality, we have $R_i = D_i$ if Q has no constraint on A_i. Let $\mathsf{Card}(T, Q)$ denote the *cardinality*, i.e., the exact number of records in T satisfying all constraints in Q. The CardEst problem requires estimating $\mathsf{Card}(T, Q)$ as accurately as possible without executing Q on T.

In this paper, we do not consider the string "LIKE" queries or pattern matching queries due to two reasons: 1) practical CardEst methods for such queries in DBMS often use magic numbers [9, 29, 41, 43], which is not meaningful to evaluate; and 2) very few

works [44, 47] use extended NLP techniques such as q-grams [30] to process such queries, thus not comparable with the mainstream CardEst solutions who can not support these queries.

CardEst Algorithms: There exist many CardEst methods in the literature, which can be classified into three classes as follows:

Traditional CardEst methods, such as histogram [42] and sampling [19, 24, 27], are widely applied in DBMS and generally based on simplified assumptions and expert-designed heuristics. Some variants of them such as [1, 11, 19, 23, 24, 45, 46, 54] are proposed later to enhance their performance.

ML-based query-driven CardEst methods try to learn a model to map each featurized query Q to its cardinality $\mathsf{Card}(T, Q)$ directly. Some ML-enhanced methods improve the performance of CardEst methods by using more complex models such as DNNs [25] or gradient boosted trees [10].

ML-based data-driven CardEst methods are independent of the queries. They regard each tuple in T as a point sampled according to the joint distribution $P_T(A) = P_T(A_1, A_2, \ldots, A_n)$. Let $P_T(Q) = P_T(A_1 \in R_1, A_2 \in R_2, \cdots, A_n \in R_n)$ be the probability specified by the region of Q. Then, we have $\mathsf{Card}(T, Q) = P_T(Q) \cdot |T|$ so CardEst problem could be reduced to model the probability density function (PDF) $P_T(A)$ of table T. A variety of ML-based models have been used in existing work to represent $P_T(A)$, the most representative of which includes deep auto-regression model [59, 61] and probabilistic graphical models such as Bayesian networks (BN) [14, 50, 56], SPN [21], and FSPN [64]. They use different techniques to balance the estimation accuracy, inference efficiency, and model size.

CardEst Benchmark: Literature works have proposed some benchmark datasets and query workloads for CardEst evaluation. We analyze their pros and cons as follows:

1) The synthetic benchmarks such as TPC-H [6] and TPC-DS [5] and Star Schema benchmarks (SSB) [36] contain real-world data schemas and synthetic generated tuples. They are mainly used for evaluating query engines but not suitable for CardEst because their data generator makes oversimplified assumptions on the joint PDF of attributes, such as uniform distribution and independence. However, real-world datasets are often highly skewed and correlated [26], which are more difficult for CardEst.

2) IMDB dataset with its JOB workload [26] is a well-recognized benchmark, containing complex data and string "LIKE" queries. However, most of the existing works [20, 25, 59, 61, 64] adjust this dataset and propose a simplified query workload JOB-LIGHT because of their inability to handle "LIKE" queries. The JOB-LIGHT workload consists of 70 realistic queries with varied number of joining tables. However, these queries touch at most 8 numerical or categorical attributes within six tables of IMDB and the join queries between these tables are only star joins centered at one table. Thus, the simplified IMDB dataset and its workload cannot comprehensively evaluate the performance of nowadays CardEst algorithms on more complex real-world data and varied join settings. On the other hand, some works [34, 59] generate queries on the IMDB dataset including "LIKE" queries which are not supported by most of recent statistical methods.

Apart from these well-established and general-purpose benchmarks, there also exist other benchmarks with specific purposes. For example, Wang [52] presents a series of real-world datasets

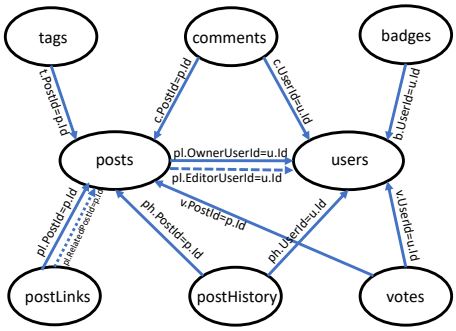

Figure 1: Join relations between tables in STATS.

to analyze whether existing CardEst algorithms are suitable to be deployed into real-world DBMS. However, it is only conducted on single-table datasets, which can not reflect the behavior of these models in more practical multi-table settings.

Summary: A surge of CardEst algorithms built on top of statistical models has been proposed in the literature, especially in the last decade. However, existing CardEst benchmarks are not sufficient to comprehensively evaluate their performance.

3 OUR NEW BENCHMARK

In this section, we design a new benchmark with complex real-world data and diverse multi-table join query workload for evaluating CardEst algorithms. To simulate practical scenarios, the benchmark should attain the following properties:

1) Large scale with enough tables, attributes, and tuples in the full outer join;

2) Complex distribution with skewed and correlated attributes whose joint distribution can not be modeled in a straightforward manner (e.g. independent assumption);

3) Rich join schema containing joins of various number of tables and diverse join forms (e.g. star and chain);

4) Diverse workload with queries covering a wide range of true cardinalities and different number of filtering and join predicates.

To this end, we establish our benchmark on a new real-world dataset with a hand-picked query workload. It overcomes the drawbacks of existing CardEst benchmarks and fully fulfills the properties listed above. We describe the details on the data and workload settings in the following content.

Data Setting: We adopt the real-world dataset STATS[1] in our benchmark. It is an anonymized dump of user-contributed content on the Stats Stack Exchange network. STATS consumes 658MB storage space with 8 tables and 71 attributes reflecting the information of users, posts, comments, tags, and their relations. A comparison of the statistical information between STATS and IMDB (the subset supporting JOB-LIGHT) is shown in Table 1. We argue that STATS more suitable for CardEst benchmark due to the following reasons:

1) Larger scale: STATS has more data tables and a larger number of numerical or categorical attributes than IMDB. Moreover, its full outer join size is four orders of magnitude larger than IMDB.

2) More complex data distribution: The distribution skewness of STATS and attribute correlation is more significant than IMDB.

[1] https://relational.fit.cvut.cz/dataset/Stats

Table 1: Comparison of IMDB and STATS dataset.

Criteria	Item	IMDB	STATS
Scale	# of tables	6	8
	# of filtering attributes	8	23
	# of attributes in per table	1-2	1-8
	full outer join size	$2 \cdot 10^{12}$	$3 \cdot 10^{16}$
Data	total attribute domain size	369, 563	578, 341
	average distribution skewness	9.159	21.798
	average pairwise correlation	0.149	0.221
Schema	join forms	star	star/chain/mixed
	# of join relations	5	12

Moreover, STATS has 3× more attributes with a larger domain size, suggesting its PDF is much harder to model than IMDB.

3) Larger query space: Each table in STATS has 1 to 8 numerical or categorical attributes that can be filtered while IMDB contains at most two in each table. Moreover, STATS's full outer join size is much larger than IMDB. These two aspects provide STATS a larger query space varying in cardinality and predicate numbers.

4) Richer join settings: The join relations between all tables in STATS are illustrated in Figure 1. IMDB contains only star joins between primary key and foreign keys (i.e. 5 join relations). Whereas, STATS has richer and more diverse join types varying in the number of joined tables (from 2 to 8), join forms (chain, star, and mixture of these two), and join keys (PK-FK and FK-FK).

In summary, item 1) and 2) make the joint PDF modeling more difficult, and item 3) and 4) make the query setting more complex and realistic. Therefore, STATS can better reveal the actual performance of CardEst algorithms in practice.

Query Workload Setting: We generate and then carefully handpick a query workload STATS-CEB on STATS to fulfill both practicality and diversity. The generation process is done in two phases.

In the first phase, we generate 70 representative join templates base on the join schema in Figure 1, each of which specifies a distinct join pattern covering a set of tables. For these join templates, we do not consider: 1) cyclic joins as most of the ML-based CardEst algorithms [20, 56, 59, 64] do not support them; and 2) non-equal joins as they rarely occur in practice and many CardEst algorithms process them in the same way as many-to-many joins. We manually check and retain each join template if it has occurred in the log data of StackExchange[2] or has its real-world semantics. To reduce redundancy, we also ensure that these join templates are not very similar (e.g. only differ in inner or outer join conditions).

In the second phrase after deriving these 70 join templates, we generate 146 queries with $1-4$ queries for each template as the testing query workload STATS-CEB. We make sure all the generated filter predicates reflect real-world semantics and diversify in multiple perspectives. In comparison to JOB-LIGHT (illustrated in Table 2), we find the following advantages of STATS-CEB:

1) More diverse queries: STATS-CEB contains twice queries as JOB-LIGHT with 3× more join templates covering a wider range of the number of joined tables.

2) Richer join types: Unlike JOB-LIGHT benchmark with on star joins, STATS-CEB contains queries with chain joins and complex mixed join forms. Moreover, JOB-LIGHT only contains queries with one-to-many PK-FK joins, whereas STATS-CEB includes queries with many-to-many FK-FK joins.

3) More filter predicates: STATS-CEB contains queries with up to 16 distinct filter predicates, which is 4× more than JOB-LIGHT.

4) Wider range of true cardinality: The cardinality range of STATS-CEB is an order of magnitude larger than JOB-LIGHT. The largest query in STATS-CEB has true cardinality of 20 billion, which is 3× larger than that of the JOB-LIGHT benchmark.

From all four aspects, we can see that STATS-CEB is a more diverse and complicated query workload than JOB-LIGHT.

Table 2: Comparison of JOB-LIGHT and STATS-CEB benchmark query workload.

Item	JOB-LIGHT	STATS-CEB
# of queries	70	146
# of joined tables	2-5	2-8
# of join templates	23	70
# of filter predicates	1-4	1-16
join type	PK-FK	PK-FK/FK-FK
true cardinality range	$9 - 9 \cdot 10^9$	$200 - 2 \cdot 10^{10}$

Summary: Our new benchmark with STATS dataset and STATS-CEB query workload has more complex data, more diverse queries, and is more close to the real-world scenarios. According to our evaluation results in the following Sections 5–7, this new CardEst benchmark helps to find more insights of existing CardEst algorithms that have not been discovered in previous works.

4 EVALUATION PLAN

We aim to evaluate how CardEst algorithms behave in a real DBMS, including the end-to-end improvement on optimizing query plans and other practicality aspects, on our new benchmark. We introduce the details of our evaluation plan in this section. In the following, Section 4.1 presents all baseline CardEst algorithms chosen to be evaluated, Section 4.2 describes our implementation method and system settings, and Section 4.3 lists the evaluation metrics of our interests.

4.1 CardEst Algorithms

Principles for Algorithm Selection and Tuning: As reviewed in Section 2, there are three classes (traditional, ML-based query-driven, and ML-based data-driven) of CardEst algorithms. We identify nine representative CardEst algorithms across the three classes. Specifically, in the class of traditional methods, we choose the simplest version of these methods, which are widely used in commercial DBMS. We do not evaluate the variants of these methods as their performance has been verified to be less competitive than ML-based methods in one or more aspects. For the ML-based methods, we choose the state-of-the-state (SOTA) methods that exhibit superiority over others. Each chosen algorithm can process multi-table join queries and support data updating.

For each CardEst algorithm, we adopt the publicly available implementation if the authors provide it and otherwise implement it by ourselves. For the hyper-parameters, if they are known to be a trade-off of some metrics, we choose the default values recommended in the original paper. Otherwise, we run a grid search to explore the combination of their values that largely improves the

[2] https://stackexchange.com

end-to-end performance on a validation set of queries. The details of each CardEst algorithm are described as follows:

Algorithm Details: For the *traditional CardEst method*, we choose the following two algorithms widely used in commericial DBMSes.

1) Histogram [42] assumes all attributes are mutually independent and maintains a 1-D (cumulative) histogram to represent $P_T(A_i)$ for each attribute A_i. The probability $P_T(Q)$ can then be easily obtained by multiplying all $P_T(R_i)$ together. It has been widely applied in DBMS such as PostgreSQL [9] and SQL Server [29]. For continuous attributes, we set its binning size to 100.

2) Sample [27, 62] makes no assumption but randomly fetches records from T on-the-fly according to $P_T(A)$ to estimate the probability $P_T(Q)$. It is also widely used in DBMS such as MySQL [41] and MariaDB [43]. We set the sampling size to 10^4.

We do not evaluate following variants of the traditional methods: multi-dimensional histogram based methods [1, 15, 32, 51] often have high storage cost; the estimation errors of kernel-based methods [19, 24] and index-based sampling techniques [27] are still high; the methods based on correcting histograms [1, 11, 23, 45], updating statistical summaries in DBMS [46, 54] and query-driven kernel-based methods[19, 24] are verified to perform much worse than the ML-based CardEst methods.

For the *ML-based query-driven CardEst methods*, we automatically generate $100K$ queries as the training examples to train these models. We choose the following most representative algorithms.

3) MSCN [25] is a deep learning method built upon the multi-set convolutional network model. The features of attributes in table T, join relations in query Q, and predicates of query Q are firstly fed into three separate modules, where each is comprised of a two-layer neural network. Then, their outputs are averaged, concatenated, and fed into a final neural network for estimation.

4) LW-XGB and 5) LW-NN [10] formulate the CardEst mapping function as a regression problem and apply gradient boosted trees and neural networks for regression, respectively. Specifically, LW-XGB applies the XGBoost [3] as it attains equal or better accuracy and estimation time than both LightGBM [22] and random forests [48] for a given model size. As the original models only support single table queries, we extend them to support joins with an additional neural network to combine single-table information.

For the *ML-based data-driven CardEst methods*, we choose four most representative and advanced methods. These methods are built upon the deep auto-regression model [59] and three probabilistic graphical models: BN [56], SPN [21], and FSPN [64].

6) NeuroCard [59], the multi-table extension of Naru [60], is built upon a deep auto-regression model. It decomposes the joint PDF $P_T(A) = P_T(A_1) \cdot \prod_{i=2}^{k} P_T(A_i|A_1, A_2, \ldots, A_{i-1})$ according to the chain rule and model each (conditional) PDF parametrically by a 4-layer DNN (4×128 neuron units). All models can be learned together using a masked auto-encoder [13]. Meanwhile, a progressive sampling technique [28] is provided to sample points from the region of query Q to estimate its probability. We set the sampling size to 8,000. We omit a very similar method in [17] as it has very close performance to NeuroCard[3].

7) BayesCard [56] is fundamentally based on BN, which models the dependence relations among all attributes as a directed acyclic graph. Each attribute A_i is assumed to be conditionally independent of the remaining ones given its parent attributes $A_{\mathrm{pa}(i)}$ so the joint PDF $\Pr_T(A) = \prod_{i=1}^{k} \Pr_T(A_i|A_{\mathrm{pa}(i)})$. BayesCard revitalizes BN using probabilistic programming to improve its inference and model construction speed (i.e., learning the dependence graph and the corresponding probability parameters). Moreover, it adopts the advanced ML techniques to process the multi-table join queries, which significantly increases its estimation accuracy over previous BN-based CardEst methods [14, 16, 50], which will not be evaluated in this paper. We use the Chow-Liu Tree [4] based method to build the structure of BayesCard and apply the complied variable elimination algorithm for inference.

8) DeepDB [21], based on sum-product networks (SPN) [7, 40], approximates $P_T(A)$ by recursively decomposing it into local and simpler PDFs. Specifically, the tree-structured SPN contains sum node to split $P_T(A)$ to multiple $P_{T'}(A)$ on tuple subset $T' \subseteq T$, product node to decompose $P_T(A)$ to $\prod_S P_T(S)$ for independent set of attributes S and leaf node if $P_T(A)$ is a univariate PDF. The SPN structure can be learned by splitting table T in a top-down manner. Meanwhile, the probability of $\Pr_T(Q)$ can be obtained in a bottom-up manner with time cost linear in the SPN's node size.

9) FLAT [64], based on factorize-split-sum-product networks (FSPN) [58], improves over SPN by adaptively decomposing $P_T(A)$ according to the attribute dependence level. It adds the factorize node to split P_T as $P_T(W) \cdot P_T(H|W)$ where H and W are highly and weakly correlated attributes in T. $P_T(W)$ is modeled in the same way as SPN. $P_T(H|W)$ is decomposed into small PDFs by the split nodes until H is locally independent of W. Then, the multi-leaf node is used to model the multivariate PDF $P_T(H)$ directly. Similar to SPN, the FSPN structure and query probability can be recursively obtained in a top-down and bottom-up fashion, respectively.

For both DeepDB and FLAT, we set the RDC thresholds to 0.3 and 0.7 for filtering independent and highly correlated attributes, respectively. Meanwhile, we do not split a node when it contains less than 1% of the input data.

Notice that, each of our evaluated CardEst algorithms is an independent and universal tool that can be easily integrated into common DBMS. There have also been proposed some CardEst modules [47, 57] that are optimized together with other components in a query optimizer in an end-to-end manner. We do not compare with them as they do not fit our evaluation framework.

4.2 Implementation and System Settings

To make our evaluation more realistic and convincing, we integrate each CardEst algorithm into the query optimizer of PostgreSQL [9], a well-recognized open-source DBMS. Then, the quality of each CardEst method can be directly reflected by the end-to-end query runtime with their injected cardinality estimation.

Before introducing the details of our integration strategy, we introduce an important concept called *sub-plan query*. For each SQL query Q, each *sub-plan query* is a query touching only a subset of tables in Q[4]. The set of all these queries is called *sub-plan query*

[3] We confirm this by email in correspondence to the authors of [17].

[4] In DBMS, sub-query refers to the nested query in the where clause, which is different for the sub-plan query concept.

space. For the example query $A \bowtie B \bowtie C$, its sub-plan query space contains queries on $A \bowtie B$, $A \bowtie C$, $B \bowtie C$, A, B, and C with the corresponding filtering predicates. The built-in planner in DBMS will generate the sub-plan query space, estimate their cardinalities, and determine the optimal execution plan. For example, the sub-plan queries A, B, and C only touch a single table, their CardEst results may affect the selection of table-scan methods, i.e. index-scan or seq-scan. The sub-plan queries $A \bowtie B$, $A \bowtie C$, and $B \bowtie C$ touch two tables. Their cardinalities may affect the join order, i.e. joining $A \bowtie B$ with C or $A \bowtie C$ with B, and the join method, i.e. nested-loop-join, merge-join, or hash-join. Therefore, the effects of a CardEst method on the final query execution plan are entirely decided by its estimation results over the sub-plan query space.

To this end, in our implementation, we overwrite the function "calc_joinrel_size_estimate" in the planner of PostgreSQL to derive the sub-plan query space for each query in the workload. Specifically, every time the planner needs a cardinality estimation of a sub-plan query, the modified function "calc_joinrel_size_estimate" will immediately capture it. Then, we call each CardEst method to estimate the cardinalities of the sub-plan queries and inject the estimations back into PostgreSQL. Afterward, we run the compiler of PostgreSQL on Q to generate the plan. It will directly read the injected cardinalities produced by each method. Finally, we execute the query with the generated plan. In this way, we can support any CardEst method without a large modification on the source code of PostgreSQL. We can report the total time (except the sub-plan space generation time) as the end-to-end time cost of running a SQL query using any CardEst method.

For the environment, we run all of our experiments in two different Linux Servers. The first one with 32 Intel(R) Xeon(R) Platinum 8163 CPUs @ 2.50GHz, one Tesla V100 SXM2 GPU and 64 GB available memory is used for model training. The other one with 64 Intel(R) Xeon(R) E5-2682 CPUs @ 2.50GHz is used for the end-to-end evaluation on PostgreSQL.

4.3 Evaluation Metrics

Our evaluation mainly focuses on *quantitative* metrics that directly reflect the performance of CardEst algorithms from different aspects. We list them as follows:

1) End-to-end time of the query workload, including both the query plan generation time and physical plan execution time. It serves as the "gold-standard" for CardEst algorithm, since improving the end-to-end time is the ultimate goal for optimizing CardEst in query optimizers. Please note that, apart from the aforementioned CardEst algorithms, we also evaluate the end-to-end time of two baselines: PostgreSQL itself and TrueCard, which injects the true cardinalities of all sub-plan queries into PostgreSQL. Therefore, TrueCard can obtain the optimal query plan and represents the best end-to-end performance a CardEst can possibly achieve.

2) Inference latency reflects the time cost for CardEst, which directly relates to the query plan generation time. It is crucial as CardEst needs to be done numerous times in optimizing the plan of each query. Specifically, an accurate CardEst method may be very time-costly in inference. Despite the fast execution time of the plans generated by this method, the end-to-end query performance can be poor because of its long plan generation time.

3) Space cost refers to the CardEst model size. A lightweight model is also desired as it is convenient to transfer and deploy.

4) Training cost refers to the models' offline training time.

5) Updating speed reflects the time cost for CardEst models update to fit the data changes. For real-world settings, this metric plays an important role as its underlying data always updates with tuples insertions and deletions.

Besides these metrics, [52] proposed some *qualitative metrics* related to the stability, usage, and deployment of CardEst algorithms and made a comprehensive analysis. Thus, we do not consider them in this paper.

In the following content, we first evaluate the overall end-to-end performance of all methods in Section 5. Then, we analyze the other practicality aspects in Section 6. At last, we point out the drawbacks of existing evaluation metric and propose a new metric as its potential substitution in Section 7.

5 HOW GOOD ARE THE CARDEST METHODS?

In this section, we first thoroughly investigate the true effectiveness of the aforementioned CardEst methods in improving query plan quality. Our evaluation focuses on a static environment where data in the system has read-only access. This setting is ubiquitous and critical for commercial DBMS, especially in OLAP workloads of data warehouses[12, 33, 38, 63]. We organize the experimental results as follows: Section 5.1 reports the overall evaluation results, Section 5.2 provides detailed analysis of the method's performance on various query types and an in-depth case study on the performance of some representative methods. Based on our observations, Section 5.3 summarizes the future research directions.

5.1 Overall End-to-End Performance

We evaluate the end-to-end performance (query execution time plus planning time) on both JOB-LIGHT and STATS-CEB benchmarks for all CardEst methods including two baselines PostgreSQL and TrueCard shown in Table 3. We also report their relative improvement over the PostgreSQL baseline as an indicator of their end-to-end performance. In the following, we will first summarize several overall observations (O) regarding Table 3, and then provide detailed analysis w.r.t. each of these CardEst methods.

O1: ML-based data-driven CardEst methods can achieve near-optimal performance, whereas traditional and ML-based query-driven CardEst methods can hardly improve over PostgreSQL on both benchmarks. The astonishing performance of these ML-based data-driven CardEst methods (BayesCard, DeepDB, and FLAT) come from their accurate characterization of data distributions and sophisticated ways of handling join queries using fanout attributes. Traditional methods (Histogram and Sample) have much worse performance than PostgreSQL because they make simplified assumptions which leads to huge estimation errors. The query-driven CardEst methods rely on a large amount of executed queries as training data and the testing query workload should follow the same distribution as the training workload in order for them to produce an accurate estimation [21]. Admittedly, the query-driven methods are more general because they can handle complex similarity queries [37, 53] and string "LIKE" queries [34, 44, 47].

Table 3: Overall performance of CardEst algorithms.

Category	Method	Workload					
		JOB-LIGHT			STATS-CEB		
		End-to-End Time	Exec. + Plan Time	Improvement	End-to-End Time	Exec. + Plan Time	Improvement
Baseline	PostgreSQL	3.67h	3.67h + 3s	0.0%	11.34h	11.34h + 25s	0.0%
	TrueCard	**3.15h**	**3.15h + 3s**	**14.2%**	**5.69h**	**5.69h + 25s**	**49.8%**
Traditional	Histogram	5.22h	5.22h + 3s	−42.2%	21.34h	21.34h + 24s	−88.2%
	Sample	4.87h	4.84h + 96s	−32.6%	> 25h	−−	−−
Query-driven	MSCN	6.21h	6.21h + 12s	−69.2%	9.13h	9.11h + 46s	19.7%
	LW-XGB	4.31h	4.31h + 8s	−17.4%	> 25h	−−	−−
	LW-NN	3.63h	3.63h + 9s	1.1%	11.33h	11.33h + 34s	0.0%
Data-driven	NeuroCard	3.41h	3.29h + 423s	6.8%	12.05h	11.85h + 709s	−6.2%
	BayesCard	3.18h	3.18h + 10s	13.3%	7.16h	7.17h + 35s	36.9%
	DeepDB	3.29h	3.28h + 33s	10.3%	6.51h	6.46h + 168s	42.6%
	FLAT	3.21h	3.21h + 15s	12.9%	5.92h	5.80h + 437s	47.8%

O2: The differences among the ML-based data-driven methods' improvements over PostgreSQL are much more drastic on datasets with more complicated data distributions and join schemas. We observe that the execution time for NeuroCard, BayesCard, DeepDB, and FLAT on JOB-LIGHT are all roughly 3.2h, which is very close to the optimal execution time (3.15h). As explained in Section 3, the data distributions in IMDB and the JOB-LIGHT queries are relatively simple. Specifically, the table title in the IMDB dataset plays a central role in the join schema that other tables are all joined with its primary key. Thus, the fanout-based data-driven methods can accurately learn the join relationship in JOB-LIGHT. However, their performance differences on STATS are very drastic because the STATS dataset is much more challenging with high attribute correlations and various join types. Therefore, the STATS-CEB benchmark helps to expose the advantages and drawbacks of these methods.

Analysis of traditional CardEst methods: Histogram and Sample perform significantly worse than PostgreSQL on both benchmarks because of their inaccurate estimation. They use the join uniformity assumption to estimate join queries, which has a fair estimation accurate for queries touching a small number of tables. The estimation error grows rapidly for queries joining more tables. These queries are generally more important in determining a good execution join order [26]. Therefore, these traditional methods based on join uniformity assumption tend to yield poor join order leading to an extremely long-run query plan. The CardEst component in PostgreSQL produces much more accurate estimation because of its high-quality implementation of histogram-based CardEst method and fine-grained optimization on join queries.

Analysis of ML-based query-driven CardEst methods: Overall the query-driven methods can not outperform the PostgreSQL baseline. Specifically, MSCN has very unstable performance (69.2% slower runtime than PostgreSQL on IMDB but 19.7% faster on STATS-CEB), LW-XGB has much slower query runtime, and LW-NN has comparable performance. The unsatisfactory performance of these methods could be due to the following reasons.

• These methods are essentially trying to fit the probability distributions of all possible joins in the schema, which has super-exponential complexity. Specifically, there can exist an exponential number possible joins in a schema, and for each joining table, the complexity of its distribution is exponential w.r.t. its attribute number and domain size [56]. Thus, these models themselves are not complex enough to fully understand all these distributions.

• Similarly, these methods would require an enormous amount of executed queries as training data to accurately characterize these complex distributions. In our experiment, our computing resources can only afford to generate $100K$ queries (executing 146 queries in STATS-CEB takes 10 hours), which may not be enough for this task. Besides, it is unreasonable to assume that a CardEst method can have access to this amount of executed queries.

• The well-known workload shift issue states that query-driven methods trained one query workload will not likely produce an accurate prediction on a different workload [21]. In our experiment, the training query workload is automatically generated whereas the JOB-LIGHT and STATS-CEB testing query workload is hand-picked. Therefore, the training and testing workload of these methods have different distributions.

Analysis of ML-based data-driven methods: Data-driven ML methods (BayesCard, NeuroCard, DeepDB, and FLAT), do consistently outperform PostgreSQL by $7-13\%$ on JOB-LIGHT. Except for NeuroCard, the other three improve the PostgreSQL by $37-48\%$. Their near-optimal performance suggests that as opposed to the traditional and query-driven CardEst methods, these SOTA data-driven methods could serve as a practical counterpart of the PostgreSQL CardEst component. Through detailed analysis of NeuroCard method, we derive the following observation.

O3: Learning one model on the (sample of) full outer join of all tables in a DB will lead to poor scalability. An effective CardEst method should make appropriate independent assumptions for large datasets. The advantage of NeuroCard over PostgreSQL disappears when shifting from JOB-LIGHT to STATS-CEB benchmark for the following reasons. First, the STATS dataset contains significantly more attributes with larger domain size, which is detrimental to NeuroCard's underlying deep autoregressive models [52, 56]. Second, the full outer join size of STATS is significantly larger than IMDB, making the sampling procedure of NeuroCard less effective. Specifically, the full outer join size can get up to 3×10^{16} and an affordable training data sample size would be no larger than 3×10^{8}. Therefore, the NeuroCard model trained on this sample only contains 1×10^{-8} of the information as the

Table 4: End-to-end time improvement ratio of CardEst algorithms on queries with different number of join tables.

# tables	# queries	MSCN	BayesCard	DeepDB	FLAT	TrueCard
2 – 3	38	2.12%	2.07%	1.98%	2.48%	**3.66%**
4	50	−92.7%	55.8%	48.0%	55.7%	**55.9%**
5	28	26.4%	36.55%	32.90%	35.4%	**37.0%**
6 – 8	34	−3.76%	2.51%	26.3%	32.0%	**34.6%**

original dataset. It can hardly capture the correct data distributions especially for join tables with small cardinalities. Specifically, we find that for queries on the joins of a small set of tables, NeuroCard's prediction deviates significantly from the true cardinality because its training sample does not contain much not-null tuples for this particular set of join tables.

All other three data-driven data-driven CardEst methods can significantly outperform the PostgreSQL baseline because their models are not constructed on the full outer join of all tables. Specifically, they all use the "divide and conquer" idea to divide the large join schema into several smaller subsets with each representing a join of multiple tables. In this way, they can capture the rich correlation within each subset of tables; simultaneously, they avoid constructing the full outer join of all tables by assuming some independence between tables with low correlations. Then, BayesCard, DeepDB, and FLAT build a model (BN, SPN, and FSPN respectively) to represent the distribution of the corresponding small part. This approach solves the drawback of NeuroCard, yields relatively accurate estimation, and produces effective query execution plans. Among them, FLAT achieves the best performance (47.8% improvement), which is near-optimal (49.8% for TrueCard). It can outperform DeepDB mostly because the STATS dataset is highly correlated, so the FSPN in FLAT has a more accurate representation of the data distribution than the SPN in DeepDB. On the other hand, BayesCard has an even more accurate representation of data distribution and yields the best end-to-end time for most queries in STATS-CEB. However, it does not outperform FLAT most because of one extremely long-run query, which we will study in detail in Section 5.2.

In the rest of this section, we will mainly carry out analysis on these ML-based data-driven CardEst methods since all other methods do not show a clear advantage over the PostgreSQL baseline.

5.2 Analysis of Different Query Settings

In this section, we further examine to what extent the CardEst methods improve over PostgreSQL on various query types, i.e. different number of join tables (*#tables*) and different intervals of true cardinalities. Since JOB-LIGHT workload does not contain queries with very diverse types and the ML-based data-driven methods do not show significant difference on these queries, we only investigate queries on STATS-CEB. Worth noticing that we only examine the methods with improvements over PostgreSQL on STATS-CEB: MSCN, BayesCard, DeepDB, and FLAT.

Number of join tables: Table 4 shows performance improvement of different ML-based methods over the PostgreSQL baseline and we derive the following observation:

O4: The improvement gap between these methods and the optimal performance (TrueCard) increase as the number of join tables increases. Specifically, we can see that BayesCard achieves near-optimal improvement for queries joining no more than 5 tables, but it barely has much improvement for queries joining 6 tables and more. This observation suggests that the estimation qualities of these SOTA methods decline for queries joining more tables. In fact, the fanout join estimation approach adopted by all these methods sacrifices accuracy for efficiency by assuming some tables are independent of others. This estimation error may accumulate for queries joining a large number of tables, leading to sub-optimal query plans.

Table 5: End-to-end time improvement ratio of CardEst algorithms on queries with different size of cardinality.

Cardinality	# queries	MSCN	BayesCard	DeepDB	FLAT	TrueCard
$0 - 10^5$	27	75.3%	83.9%	71.6%	74.3%	**89.3%**
$10^5 - 10^7$	74	−60.2%	19.4%	−27.7%	−7.7%	**26.3%**
$10^7 - 10^8$	25	0.03%	23.31%	2.16%	0.77%	**30.3%**
$10^8 - 10^9$	14	−56.1%	2.02%	−11.4%	7.00%	**10.4%**
$10^9 - 10^{10}$	3	9.15%	29.7%	1.15%	29.6%	**31.0%**
$10^{10} - 10^{11}$	3	23.5%	40.1%	51.6%	**53.8%**	**53.9%**

Size of cardinality: Table 5 reports these ML-based methods' performance improvement for queries with different true cardinality size. Specifically, DeepDB shows unsatisfactory performance on queries with small cardinality (less than 10^{10}) and only outperforms the baseline for three queries with the largest cardinalities. However, DeepDB's overall performance on STATS-CEB still beats BayesCard, which exhibits constantly better performance on all intervals except the largest one. This observation suggests that queries with larger cardinality can have a much longer runtime and thus dominate the overall performance. We provide the following detail case study to investigate this observation.

We choose Q57 (in Figure 2) of STATS-CEB as a representative query to thoroughly study when a CardEst method could go wrong. The execution time of Q57 for TrueCard and FLAT is $1.92h$ and $1.94h$, while the time for BayesCard is $3.23h$. We derive two important observations from this query, which are verified to be generalizable to other queries in JOB-LIGHT and STATS-CEB.

O5: Accurate estimation of (sub-plan) queries with large cardinalities is much more important than the small ones. When choosing the join method in the root node of execution plans for Q57, BayesCard underestimates the final join size and chooses the "merge join" physical operation. Alternatively, FLAT produces a more accurate estimation for the final join size and chooses the "hash join" operation, which is twice faster as the "merge join". Since the final join operation takes up 99% of the total execution time, FLAT significantly outperforms BayesCard on this query.

Generally, the (sub-plan) query with larger true cardinality requires a longer time to execute. It is very common that the join size of two intermediate tables is much larger than both of them. Therefore, some sub-plans can take a significantly longer time to execute than other sub-plans. A bad estimation on these large sub-plan queries can have a detrimental result on the overall runtime, whereas a series of good estimations on small sub-plan queries will not influence the runtime as much. Therefore, the estimation accuracy of sub-plan queries with very large true cardinalities dominate the overall quality of the query plan.

O6: Choosing the correct physical operations sometimes is more important than selecting the optimal join order. As shown in Figure 2 BayesCard can generate the optimal join order of Q57 because of its near-perfect estimation of all sub-plan

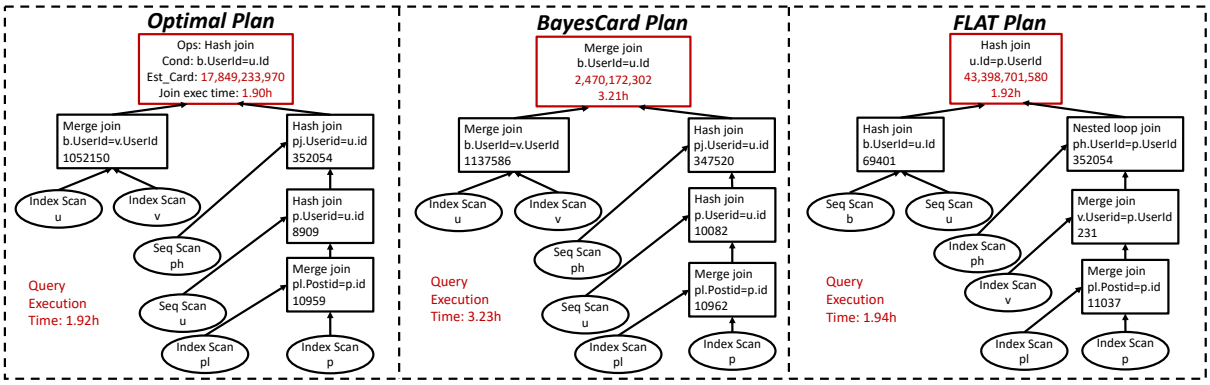

Figure 2: Case study of STATS Q57.

queries except for the one at the root node. The join order selected by FLAT is very different from the optimal one. Surprisingly, FLAT's plan is roughly twice faster to execute than BayesCard's plan due to the dominant large sub-plan query at the root node. For Q57, a sub-optimal query plan is only 1% slower to execute but only one sub-optimal physical operation is 68% slower.

These aforementioned two observations also hold for other queries in these two benchmarks, so we conjecture that they might be generalizable to all queries.

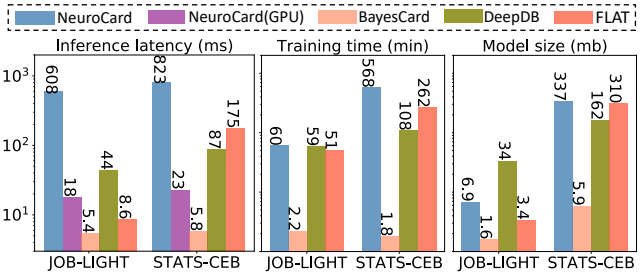

Figure 3: Practicality aspects of CardEst algorithms.

5.3 Future Research Directions

Based on the exhaustive analysis in this section, we point out several future research directions (RD).

• **RD1**: Based on the important observation in Section 5.2, we advocate researchers investigate and develop CardEst method that can produce accurate estimation for queries with large cardinalities instead of fine-grained estimation on extremely small ones.

• **RD2**: The bottleneck of ML-based query-driven methods is that they require an impractical amount of time to generate training queries, which maybe ineffective if the future queries do not follow the same distribution. Therefore, one particularly important research direction is to improve the effectiveness and generating efficiency of training data, such as using AQP or meta-learning, suggested by recent researches [34, 57].

• **RD3**: For the ML-based data-driven methods, researchers should focus on improving the "divide and conquer" fanout methods for join queries used by DeepDB, BayesCard, and FLAT instead of building one model on the full outer join of all tables.

• **RD4**: These ML-based data-driven methods currently can not support complex queries with "LIKE" filter predicates and cyclic joins. Therefore, equipping these methods with techniques to process these queries can be crucial to enhance their applicability.

6 WHAT OTHER ASPECTS OF CARDEST METHODS MATTER?

In addition to CardEst's improvement in execution time, we discuss model practicality aspects in this section: inference latency (in Section 6.1), model size and training time (in Section 6.2), and model update speed and accuracy (in Section 6.3). We only compare the recently proposed ML-based data-driven CardEst methods, DeepDB, NeuroCard, BayesCard, and FLAT, which have been proved to significantly improve the Postgres baseline.

6.1 Inference Latency

The end-to-end query time is comprised of query execution and planning time, the latter of which is essentially determined by the CardEst method's inference latency. Commercial DBMS normally has a negligible planning time due to their simplified cardinality estimator and engineering effort to accelerate the inference speed. However, the inference latency of some ML-based data-driven methods can approach one second per sub-plan query, which slows down the end-to-end query execution time.

O7: Inference latency has a non-trivial impact on the end-to-end query time. Specifically, revisiting Table 3, we observe that the pure execution time for NeuroCard on the JOB-LIGHT benchmark is near-optimal but the overall runtime is significantly longer than FLAT and BayesCard because of its slow inference procedure on CPU. Admittedly, for some extremely long-run queries (e.g. query 57 of STATS-CEB), inference latency is overshadowed by the execution time. However, for most queries, inference latency does have a large impact on the overall end-to-end query time because planning one query will request the CardEst methods to estimate dozens of its sub-plan queries. Therefore, we claim that a CardEst method is not very practical if it takes more than couple hundreds milliseconds to estimate one sub-plan query.

Figure 3 reports the average inference latencies of all sub-queries in the workload for each method. Their inference speed can be ranked as BayesCard > NeuroCard(GPU) > FLAT/DeepDB >> NeuroCard. The newly proposed inference algorithms on BN provide BayesCard with a very fast and stable inference speed on both benchmarks. However, the inference speeds of FLAT and DeepDB

are not as stable because they tend to build much larger models with more computation circuits for the more complicated database STATS. The inference process of NeuroCard requires a large number of progressive samples and its underlying DNN is computationally demanding on CPUs. Therefore, we observe that the inference speed is greatly improved by running it on GPUs.

6.2 Model Deployment

While the JOB-LIGHT benchmark can not distinguish these methods' training time and model size, the advantages and drawbacks of these methods are apparent on STATS-CEB. Based on Figure 3, we derive the following observation.

O8: BN-based CardEst approach is very friendly for system deployment. First of all, the BN-based approaches, such as BayesCard, are generally interpretable and predictable, thus easy to debug for DBMS analytics. More importantly, a CardEst method friendly for system deployment should have faster training time and lightweight model size and BayesCard has the dominant advantage over the other ML-based data-driven methods in these two aspects because of its underlying Bayesian model. Specifically, from both training time and model size aspects, these methods can be ranked as BayesCard << FLAT/DeepDB < NeuroCard. We provide the detailed reasoning as follows.

BayesCard proposes an accelerated model construction process of BN using probabilistic programming. Its model training time is roughly 100 times faster than the other three data-driven methods. Moreover, the BNs in BayesCard, which utilize the attribute conditional independence to reduce the model redundancy, are naturally compact and lightweight.

FLAT and DeepDB recursively learn the underlying FSPN and SPN models. Their training time is less stable and varies greatly with the number of highly correlated attributes in the datasets. Thus, we observe a much longer training time on STATS than on the IMDB dataset for these two methods. The SPNs in DeepDB iteratively split the datasets into small regions, aiming to find local independence between attributes within each region. However, in presence of highly correlated attributes (e.g. STATS), the SPNs tend to generate a long chain of dataset splitting operation, leading to long training time and a very large model size. The FSPNs in FLAT effectively address this drawback of SPNs by introducing the factorize operation but their training time and model size suffer greatly for datasets with a large number of attributes (e.g. STATS) because of the recursive factorize operations.

The training of NeuroCard is particularly long and its size is also the largest on STATS because its join schema does not form a tree and NeuroCard currently only supports tree-structured schema. Thus, we extract 16 unique tree sub-structures from its schema graph and train one NeuroCard model on each tree to handle the STATS-CEB queries. Therefore, we argue that extending NeuroCard for non-tree-structured schemas can greatly improve its practicality.

6.3 Model Update

Model update is a crucial aspect when deploying a CardEst method in OLTP databases. Frequent data updates in these DBs require the underlying CardEst method to swiftly update itself and adjust to the new data accurately. In the following, we first provide our observation regarding the updatability of ML-based query-driven methods and then provide the update experimental settings and results for ML-based data-driven methods on the STATS dataset.

Table 6: Update performance of CardEst algorithms.

Criteria	NeuroCard	BayesCard	DeepDB	FLAT
Update time	5,569s	**12s**	248s	360s
Original E2E time (Table 3)	11.85h	7.16h	6.46h	5.80h
E2E time after update	13.94h	7.16h	6.72h	7.04h

O9: Existing query-driven CardEst methods are impractical for dynamic DBs. The query-driven models require a large amount of executed queries to train their model, which might be unavailable for a new DB and very time-consuming to generate (e.g. 146 STATS-CEB queries take more than ten hours to execute). More importantly, they need to recollect and execute the queries whenever datasets change or query workload shifts. Therefore, they can not keep up with the frequent data update in dynamic DBs.

Experimental settings: To simulate a practical dynamic environment, we split the STATS data into two parts based on the timestamps of tuples. We first train a stale model for each method on the data created before 2014 (roughly 50%) and insert the rest of the data to update these models. We only test the data insertion scenario since some methods (NeuroCard and DeepDB) do not support data deletion. We use the update algorithm in these methods' publicly available source code.

Experimental results: As shown in Table 6, we record the time these methods take to update their stale models and evaluate the end-to-end query performance of the updated models on STATS-CEB queries. We also cite the original model performance from Table 3 as comparison baselines. We first summarize the most important observation based on this table and then provide detailed reasonings from the update speed and accuracy perspectives.

O10: Data-driven CardEst methods have the potential to keep up with fast data update and can be applied in dynamic DBs. Specifically, BayesCard takes 12*s* to update itself for an insertion of millions of tuples on multiple tables in the DB. More important, its end-to-end performance is unaffected by this massive update, thus very suitable for dynamic DBs.

Update speed of these methods can be ranked as BayesCard >> DeepDB > FLAT > NeuroCard. BayesCard preserves its underlying BN's structure and only incrementally updates the model parameters. Since its model size is relatively small, the update speed is more than 20 times faster than others. DeepDB and FLAT also preserves their underlying structure of SPN and FSPN but as their structures are significantly larger, the incrementally updating their model parameters still take a large amount of time.

Update accuracy can be ranked as BayesCard > DeepDB > FLAT > NeuroCard. BayesCard's underlying BN's structure captures the inherent causality, which is unlike to change when data changes. Therefore, BayesCard can preserve its original accuracy after model update (i.e. same as its comparison baseline). The structures of SPN in DeepDB and FSPN in FLAT are learned to fit the data before the update and cannot extrapolate well to the newly inserted data. Therefore, only updating the model parameters will cause modeling inaccuracy (i.e. we observe a drop in their end-to-end performance when compared with their baselines).

Table 7: Comparison between Q-Error and P-Error.

JOB-LIGHT Workload								STATS-CEB Workload							
Method	Execution Time (h)	Q-Error			P-Error			Method	Execution Time (h)	Q-Error			P-Error		
	(Descending Order)	50%	90%	99%	50%	90%	99%		(Descending Order)	50%	90%	99%	50%	90%	99%
MSCN	3.50	1.724	12.684	182.359	1.126	2.805	9.747	Sample	> 25	5.754	466.6	$3 \cdot 10^5$	1.139	4.651	536.3
Histogram	5.22	6.811	295.4	$4 \cdot 10^4$	1.134	3.194	20.82	LW-XGB	> 25	5.652	453.2	$3 \cdot 10^5$	1.742	6.627	526.2
Sample	4.84	3.259	135.4	$6 \cdot 10^4$	1.000	1.345	3.593	Histogram	21.34	10.34	439.4	$3 \cdot 10^5$	1.333	13.12	497.3
LW-XGB	4.31	5.303	261.2	$2 \cdot 10^5$	1.0	3.728	15.48	NeuroCard	11.85	951.4	$9 \cdot 10^5$	$6 \cdot 10^8$	1.193	2.844	$1 \cdot 10^3$
PostgreSQL	**3.67**	**1.810**	**8.089**	**51.80**	**1.000**	**1.408**	**2.329**	**PostgreSQL**	**11.34**	**1.439**	**11.08**	$\mathbf{2 \cdot 10^3}$	**1.000**	**1.595**	**12.35**
LW-NN	3.63	5.713	112.9	$4 \cdot 10^4$	1.104	2.448	43.67	LW-NN	11.33	15.73	832.9	$2 \cdot 10^4$	1.159	4.651	18.02
NeuroCard	3.29	2.153	10.17	26.89	1.011	1.984	2.747	MSCN	9.11	24.19	510.5	$1 \cdot 10^4$	1.188	4.466	11.11
DeepDB	3.28	1.332	2.257	33.29	1.000	1.061	1.528	BayesCard	7.16	1.182	50.40	156.4	1.000	1.582	6.843
FLAT	3.21	1.230	1.987	15.71	1.000	1.062	1.528	DeepDB	6.46	2.451	22.37	$1 \cdot 10^3$	1.030	1.833	6.819
BayesCard	3.18	1.145	2.835	10.33	1.000	1.002	1.123	FLAT	5.80	1.675	10.44	768.8	1.000	1.346	5.546

7 IS CURRENT METRIC GOOD ENOUGH?

Most of the existing researches [10, 20, 25, 59, 64] use the Q-Error metric [31] to evaluate the quality of their proposed CardEst method's. Since the ultimate goal of CardEst methods to assist the optimizer to generate better query execution plans, we explore whether the Q-Error metric can directly reflect the query execution time.

We first compare the Q-Error and execution time for each CardEst method on both benchmarks and show that better Q-Error does not necessarily lead to shorter query execution time (in Section 7.1). Then, we identify the limitations of Q-Error and propose another metric(P-Cost) to address these limitations. At last, we show that P-Error has better correspondence to the query execution and advocate it to be a potential substitution of Q-Error (in Section 7.2).

7.1 Problems with Q-Error

Q-Error is a well-known metric to evaluate the quality of different CardEst methods. It measures the deviations of the estimated cardinality of a CardEst method from the actual one, defined as:

$$\textbf{Q-Error} = max(\frac{\text{Estimated Cardinality}}{\text{True Cardinality}}, \frac{\text{True Cardinality}}{\text{Estimated Cardinality}}).$$

Obviously, perfect Q-Error (i.e. Q-Error = 1) suggests that the estimation equals the true cardinality and thus, CardEst method with perfect Q-Error on all sub-plan queries should yield the optimal result. However, it remains to investigate whether an imperfect Q-Error metric can directly reflect the query execution time. We report the distributions (50%, 90%, 99% percentiles) of all sub-plan queries' Q-Errors for both benchmarks in Table 7, where we sort all CardEst methods in descending order of their execution time. Please note that Q-Error for these methods reported on JOB-LIGHT is different than the one reported in their original paper because here we focus on all sub-plan queries instead of the JOB-LIGHT queries themselves. We derive the following observation:

O11: The Q-Error metric can not serve as a good indicator for query execution performance. This observation is supported by a large amount of evidence for queries on the JOB-LIGHT and STATS-CEB benchmark. We enumerate four examples below: 1) MSCN on STATS-CEB shows significantly worse Q-Error on all distribution percentiles than the PostgreSQL baseline, but its query plan still outperforms the baseline by 18%; 2) LW-XGB and LW-NN have comparable Q-Error, yet the effectiveness of their query plans on both benchmarks are drastically different; 3) NeuroCard on STATS-CEB has the worst Q-Error among over CardEst methods, but its performance is comparable to PostgreSQL and much better than the traditional methods (Histogram and Sample); and 4) BayesCard on STATS-CEB has the best Q-Error, yet execution time is 1.4h slower than FLAT.

This surprising results bring a confusion: *since better Q-Error distributions suggest better overall estimation accuracy, how does more accurate estimation sometimes lead to a worse execution plan?* The investigation of this confusion is particularly important. The DB and ML research communities have made great efforts in designing CardEst methods to improve the Q-Error metric of estimation. Most of these works all adopt the methodology that the problem CardEst is to minimize the Q-Error but neglect the true function of CardEst in a real DBMS. The observation *O9* may suggest some backfire of what recent researches trying to achieve if better Q-Error can not guarantee to improve DBMS query optimizers. Based on our thorough analysis of each query in the benchmarks, we point out two observations regarding the intrinsic problems of Q-Error.

O12: Q-Error does not distinguish queries with small and large cardinality. Q-Error only measures the relative multiplicative error, so an estimation 1 for true cardinality 10 has the same Q-Error as an estimation 10^{11} for true cardinality 10^{12}. The previous case may barely affect the overall query plan, whereas the latter one can be catastrophic because as explained in the previous section, (sub-plan) queries with large cardinalities dominate the overall effectiveness of the query plan.

For example in Q57 of Section 5.2, BayesCard has better Q-Error distribution on the sub-plan queries of Q57 than FLAT but it fails to correctly estimate the largest one and leads to a much slower execution plan. One can not expect this result by purely comparing the Q-Error distributions of these two methods.

This observation also helps to explain why NeuroCard with 100× worse Q-Error than the traditional methods can generate a much faster query execution plan. As explained in the previous section, NeuroCard learned on a small sample of the full outer join can not accurately estimate the queries touching a small number of tables (i.e. usually small cardinalities). Thus, it generates a larger Q-Error on these queries with relatively small cardinalities. On the other hand, the traditional methods using join uniformity assumptions, yield huge Q-Error queries joining a large number of tables (i.e. usually large cardinalities). Therefore, despite the overall worse Q-Error of NeuroCard, it can still outperform the traditional ones.

O13: Q-Error can not distinguish between query underestimation and overestimation. Q-Error only describes the estimate deviations from true cardinality, so an underestimation 10^9 for true cardinality 10^{10} has the same Q-Error as an overestimation 10^{11}.

These two estimations are very likely to lead to different plans with drastically different execution time. For example in Q57, BayesCard underestimates the cardinality of the final sub-plan query by 7× and selects a "merge join" operation. We test this query but injecting a 7× overestimation for this sub-plan query, and it selects the optimal "hash join" operation, which runs twice faster.

7.2 An Alternative Metric: P-Error

Obviously, the best way to evaluate the quality of a CardEst method is to directly compute its query execution time on some benchmark datasets and query workloads (e.g. JOB-LIGHT and STATS-CEB). However, computing the runtime of queries can take hours. Thus, it is not suitable for situations where fast evaluation is needed, such as hyper-parameter tuning, for which one would like to search a large number of hyper-parameter combinations and take the one with the best performance on a validation set of queries. Therefore, it is necessary to have a CardEst evaluation metric that can be computed as fast as Q-Error and simultaneously has a better corresponding to query runtime than Q-Error. In the following, we first introduce the P-Error metric and then quantitatively demonstrate that P-Error can be a possible substitute for Q-Error.

P-Error metric for CardEst: A recent research [34] also spotted some weaknesses of Q-Error as an optimization target and proposed a new optimization target plan cost (P-cost) as an alternative for training query-driven CardEst method. Based on the plan cost, we propose a new evaluation metric P-Error.

The plan cost P-cost(A, Q) refers to the cost (estimated by a DBMS) of Q's query plan generated by a CardEst method A. Since the cost model of modern DBMS is very accurate [26], we believe that P-cost should reflect the actual query execution time better. In our experiment, we calculate the PostgreSQL plan cost $PPC(A, Q)$ of each CardEst method A on each query Q for both benchmarks, which can be efficiently derived using the "EXPLAIN" operation without actually executing Q. On the other hand, we inject the true cardinalities of all sub-plan queries into PostgreSQL and let it search for the optimal query plan. We denote the cost of this optimal plan as $OPT(Q)$. Similar to Q-Error, our P-Error metric is also defiend as a ratio: **P-Error** $= PPC(A, Q)/OPT(Q)$. Clearly, the value of $PPC(A, Q)$ can not be smaller than the $OPT(Q)$ so **P-Error** ≥ 1. Note that, for a validation set of queries, one can precomputed $OPT(Q)$ offline and store the value. Therefore, P-Error can be computed as efficiently as Q-Error.

Quanlity measure of a CardEst metric: Observe that when selecting the hyper-parameters or essentially comparing two CardEst methods, what really matters is the performance rank instead of the performance difference between these methods. Therefore, we can quantitatively measure how good is a CardEst metric as how often does a better metric value suggests a faster execution time for two CardEst methods. Formally, given an evaluation metric M, two CardEst methods A and B, and a workload of n queries $Q_1, Q_2, \ldots, Q_n$, we denote $M(A, Q_i)$ as the metric value of A on query Q_i (assuming smaller value indicating better performance) and $ET(A, Q_i)$ as the execution time by injecting A's estimated cardinalities into a DBMS (PostgreSQL). We denote $\mathbf{1}_{Q_i}$ as the event that $(M(A, Q_i) \leq M(B, Q_i) \wedge ET(A, Q_i) \leq ET(B, Q_i)) \vee (M(A, Q_i) \geq M(B, Q_i) \wedge ET(A, Q_i) \geq ET(B, Q_i))$. The CardEst Metric Measurement (CEMM) of M is defined as: $CEMM(M) = \sum_{i=1}^{n} \mathbf{1}_{Q_i}/n$.

O14: P-Error has better correspondence to the query execution time than Q-Error. We report the distributions (50%, 90%, 99% percentiles) of all CardEst methods on queries in both benchmarks in Table 7. We can roughly see that methods with better runtime tend to have smaller P-Error (e.g. FLAT has the best P-Error).

For a more detailed comparison between P-Error and Q-Error, we compute the average CEMM of both metrics when comparing the ML-based data-driven methods NeuroCard, BayesCard, DeepDB, and FLAT on STATS-CEB. Note that the Q-Error metric is defined as an aggregation (e.g. median) on all sub-plan queries of a query Q_i. On one hand, the CEMM(Q-Error) is close to 0.6, indicating that Q-Error can not correctly distinguish the performance of these methods since the CEMM of a completely random metric is close to 0.5. On the other hand, the CEMM(P-Error) is 0.85, indicating a better correspondence to the query execution time than Q-Error.

8 DISCUSSIONS AND CONCLUSIONS

In this paper, we establish a new benchmark for CardEst, which contains the complex real-world dataset STATS and the diverse query workload STATS-CEB. This new benchmark helps to clearly identify the pros and cons of different CardEst methods. In addition, we propose the new metric P-Error as a potential substitute for the well-known Q-Error. Based on the exhaustive experimental analysis, we derive a series of important observations that will provide the DBMS community with a holistic view of the CardEst problem and help researchers design more effective and efficient CardEst methods. At last, we summarize the following key takeaway messages and future research opportunities:

• **Overall performance:** ML-based data-driven CardEst methods can achieve near-optimal performance, whereas the other methods barely have any improvement over PostgreSQL. Admittedly, the query-driven methods are more general because they can handle complex string "LIKE" queries.

• **Importance of different queries:** Accurate estimation of queries with large cardinalities is much important than the small ones. Therefore, researchers should develop CardEst methods that can produce accurate estimation for queries with large cardinalities instead of fine-grained estimation on extremely small ones.

• **Estimation of multi-table join queries:** The ML-based methods exhibit degrading performance for queries with an increasing number of join tables. Since learning one large data-driven model on the (sample of) full outer join of all tables has poor scalability, we believe an effective CardEst method should make appropriate independent assumptions and advocate researchers follow and improve the fanout methods first proposed by DeepDB [20].

• **Practicality aspects:** The inference latency of CardEst methods has a non-trivial impact on the end-to-end query time. Existing ML-based query-driven methods are inherently impractical for dynamic DBs with frequent data updates. Therefore, designing CardEst methods with fast inference speed and effective update algorithms is also very important.

• **Problems of Q-Error:** The well-recognized Q-Error metric does not reflect a CardEst method's end-to-end query performance. Alternatively, the newly proposed P-Error metric has better correspondence to the query performance and could serve as a better optimization objective for future researches.

REFERENCES

[1] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2001. STHoles: a multidimensional workload-aware histogram. In *SIGMOD*. 211–222.

[2] Surajit Chaudhuri, Vivek Narasayya, and Ravi Ramamurthy. 2009. Exact cardinality query optimization for optimizer testing. *Proceedings of the VLDB Endowment* 2, 1 (2009), 994–1005.

[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[4] C. Chow and Cong Liu. 1968. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory* 14, 3 (1968), 462–467.

[5] Transaction Processing Performance Council(TPC). 2021. TPC-DS Vesion 2 and Version 3. *http://www.tpc.org/tpcds/* (2021).

[6] Transaction Processing Performance Council(TPC). 2021. TPC-H Vesion 2 and Version 3. *http://www.tpc.org/tpch/* (2021).

[7] Mattia Desana and Christoph Schnörr. 2020. Sum-product graphical models. *Machine Learning* 109, 1 (2020), 135–173.

[8] Amol Deshpande, Minos Garofalakis, and Rajeev Rastogi. 2001. Independence is good: Dependency-based histogram synopses for high-dimensional data. *ACM SIGMOD Record* 30, 2 (2001), 199–210.

[9] Postgresql Documentation 12. 2020. Chapter 70.1. Row Estimation Examples. *https://www.postgresql.org/docs/current/row-estimation-examples.html* (2020).

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

[11] Dennis Fuchs, Zhen He, and Byung Suk Lee. 2007. Compressed histograms with arbitrary bucket layouts for selectivity estimation. *Information Sciences* 177, 3 (2007), 680–702.

[12] Hector Garcia-Molina and Gio Wiederhold. 1982. Read-only transactions in a distributed database. *ACM Transactions on Database Systems (TODS)* 7, 2 (1982), 209–234.

[13] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. 2015. MADE: Masked autoencoder for distribution estimation. *International Conference on Machine Learning* (2015), 881–889.

[14] Lise Getoor, Benjamin Taskar, and Daphne Koller. 2001. Selectivity estimation using probabilistic models. In *SIGMOD*. 461–472.

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

[16] Max Halford, Philippe Saint-Pierre, and Franck Morvan. 2019. An approach based on bayesian networks for query selectivity estimation. *DASFAA* 2 (2019).

[17] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2019. Multi-attribute selectivity estimation using deep learning. In *SIGMOD*.

[18] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1035–1050.

[19] Max Heimel, Martin Kiefer, and Volker Markl. 2015. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*. 1477–1492.

[20] Benjamin Hilprecht. 2019. Github repository: deepdb public. *https://github.com/DataManagementLab/deepdb-public* (2019).

[21] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2019. DeepDB: learn from data, not from queries!. In *PVLDB*.

[22] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017), 3146–3154.

[23] Andranik Khachatryan, Emmanuel Müller, Christian Stier, and Klemens Böhm. 2015. Improving accuracy and robustness of self-tuning histograms by subspace clustering. *IEEE TKDE* 27, 9 (2015), 2377–2389.

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

[25] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2019. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR*.

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

[27] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. 2017. Cardinality Estimation Done Right: Index-Based Join Sampling. In *CIDR*.

[28] Eric Liang, Zongheng Yang, Ion Stoica, Pieter Abbeel, Yan Duan, and Peter Chen. 2020. Variable Skipping for Autoregressive Range Density Estimation. In *ICML*. 6040–6049.

[29] Pedro Lopes, Craig Guyer, and Milener Gene. 2019. Sql docs: cardinality estimation (SQL Server). *https://docs.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver15* (2019).

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *NIPS* (2013).

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

[32] M Muralikrishna and David J DeWitt. 1988. Equi-depth multidimensional histograms. In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*. 28–36.

[33] Yoon-Min Nam Nam, Donghyoung Han Han, and Min-Soo Kim Kim. 2020. SPRINTER: a fast n-ary join query processing method for complex OLAP queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2055–2070.

[34] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *arXiv preprint arXiv:2101.04964* (2021).

[35] Parimarjan Negi, Ryan Marcus, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2020. Cost-guided cardinality estimation: Focus where it matters. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 154–157.

[36] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The star schema benchmark and augmented fact table indexing. In *Technology Conference on Performance Evaluation and Benchmarking*. Springer, 237–252.

[37] Yeonsu Park, Seongyun Ko, Sourav S Bhowmick, Kyoungmin Kim, Kijae Hong, and Wook-Shin Han. 2020. G-CARE: A Framework for Performance Benchmarking of Cardinality Estimation Techniques for Subgraph Matching. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1099–1114.

[38] Pedro Pedreira, Yinghai Lu, Sergey Pershin, Amit Dutta, and Chris Croswhite. 2018. Rethinking concurrency control for in-memory OLAP dbmss. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1453–1464.

[39] Matthew Perron, Zeyuan Shang, Tim Kraska, and Michael Stonebraker. 2019. How I learned to stop worrying and love re-optimization. In *ICDE*. 1758–1761.

[40] Hoifung Poon and Pedro Domingos. 2011. Sum-product networks: A new deep architecture. In *ICCV Workshops*. 689–690.

[41] MySQL 8.0 Reference Manual. 2020. Chapter 15.8.10.2 Configuring Non-Persistent Optimizer Statistics Parameters. *https://dev.mysql.com/doc/refman/8.0/en/innodb-statistics-estimation.html* (2020).

[42] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1979. Access path selection in a relational database management system. In *SIGMOD*. 23–34.

[43] MariaDB Server Documentation. 2020. Statistics for optimizing queries: InnoDB persistent statistics. *https://mariadb.com/kb/en/innodb-persistent-statistics/* (2020).

[44] Suraj Shetiya, Saravanan Thirumuruganathan, Nick Koudas, and Gautam Das. 2020. Astrid: accurate selectivity estimation for string predicates using deep learning. *Proceedings of the VLDB Endowment* 14, 4 (2020), 471–484.

[45] Utkarsh Srivastava, Peter J Haas, Volker Markl, Marcel Kutsch, and Tam Minh Tran. 2006. Isomer: Consistent histogram construction using query feedback. In *ICDE*. 39–39.

[46] Michael Stillger, Guy M Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO-DB2's learning optimizer. In *PVLDB*, Vol. 1. 19–28.

[47] Ji Sun and Guoliang Li. 2019. An end-to-end learning-based cost estimator. *VLDB* (2019).

[48] Vladimir Svetnik, Andy Liaw, Christopher Tong, J Christopher Culberson, Robert P Sheridan, and Bradley P Feuston. 2003. Random forest: a classification and regression tool for compound classification and QSAR modeling. *Journal of chemical information and computer sciences* 43, 6 (2003), 1947–1958.

[49] Immanuel Trummer. 2019. Exact cardinality query optimization with bounded execution cost. In *Proceedings of the 2019 International Conference on Management of Data*. 2–17.

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

[51] Hai Wang and Kenneth C Sevcik. 2003. A multi-dimensional histogram for selectivity estimation and fast approximate query answering. In *Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*. 328–342.

[52] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *VLDB* 14, 9 (2021), 1640–1654.

[53] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Xin Cao, Yifang Sun, Wei Wang, and Makoto Onizuka. 2020. Monotonic cardinality estimation of similarity selection: A deep learning approach. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1197–1212.

[54] Chenggang Wu, Alekh Jindal, Saeed Amizadeh, Hiren Patel, Wangchao Le, Shi Qiao, and Sriram Rao. 2018. Towards a learning optimizer for shared clouds. *PVLDB* 12, 3 (2018), 210–222.

[55] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*.

[56] Ziniu Wu and Amir Shaikhha. 2020. BayesCard: A Unified Bayesian Framework for Cardinality Estimation. *arXiv preprint arXiv:2012.14743* (2020).

[57] Ziniu Wu, Peilun Yang, Pei Yu, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2021. A Unified Transferable Model for ML-Enhanced DBMS. *arXiv preprint arXiv:2105.02418* (2021).

[58] Ziniu Wu, Rong Zhu, Andreas Pfadler, Yuxing Han, Jiangneng Li, Zhengping Qian, Kai Zeng, and Jingren Zhou. 2020. FSPN: A New Class of Probabilistic Graphical Model. *arXiv preprint arXiv:2011.09020* (2020).

[59] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2021. NeuroCard: One Cardinality Estimator for All Tables. *PVLDB* 14, 1 (2021), 61–73.

[60] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep unsupervised cardinality estimation. *PVLDB* (2019).

[61] Zongheng Yang and Chenggang Wu. 2019. Github repository: naru project. *https://github.com/naru-project/naru* (2019).

[62] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random sampling over joins revisited. In *SIGMOD*. 1525–1539.

[63] Zhuoyue Zhao, Feifei Li, and Yuxi Liu. 2020. Efficient join synopsis maintenance for data warehouse. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2027–2042.

[64] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *VLDB* 14, 9 (2021), 1489–1502.