Robust Query Driven Cardinality Estimation under Changing Workloads

Parimarjan Negi
MIT CSAIL
pnegi@mit.edu

Ziniu Wu
MIT CSAIL
ziniuw@mit.edu

Andreas Kipf
MIT CSAIL
kipf@mit.edu

Nesime Tatbul
MIT CSAIL, Intel Labs
tatbul@csail.mit.edu

Ryan Marcus
MIT CSAIL, Intel Labs
rcmarcus@mit.edu

Sam Madden
MIT CSAIL
madden@csail.mit.edu

Tim Kraska
MIT CSAIL
kraska@csail.mit.edu

Mohammad Alizadeh
MIT CSAIL
alizadeh@csail.mit.edu

ABSTRACT

Query driven cardinality estimation models learn from a historical log of queries. They are lightweight, having low storage requirements, fast inference and training, and are easily adaptable for any kind of query. Unfortunately, such models can suffer unpredictably bad performance under workload drift, i.e., if the query pattern or data changes. This makes them unreliable and hard to deploy. We analyze the reasons why models become unpredictable due to workload drift, and introduce modifications to the query representation and neural network training techniques to make query-driven models robust to the effects of workload drift. First, we emulate workload drift in queries involving some unseen tables or columns by randomly masking out some table or column features during training. This forces the model to make predictions with missing query information, relying more on robust features based on up-to-date DBMS statistics that are useful even when query or data drift happens. Second, we introduce join bitmaps, which extends sampling-based features to be consistent across joins using ideas from sideways information passing. Finally, we show how both of these ideas can be adapted to handle data updates.

We show significantly greater generalization than past works across different workloads and databases. For instance, a model trained with our techniques on a simple workload (JOBLight-train), with $40k$ synthetically generated queries of at most 3 tables each, is able to generalize to the much more complex Join Order Benchmark, which include queries with up to 16 tables, and improve query runtimes by 2× over PostgreSQL. We show similar robustness results with data updates, and across other workloads. We discuss the situations where we expect, and see, improvements, as well as more challenging workload drift scenarios where these techniques do not improve much over PostgreSQL. However, even in the most challenging scenarios, our models never perform worse than PostgreSQL, while standard query driven models can get much worse than PostgreSQL.

PVLDB Reference Format:
Parimarjan Negi, Ziniu Wu, Andreas Kipf, Nesime Tatbul, Ryan Marcus, Sam Madden, Tim Kraska, and Mohammad Alizadeh. Robust Query Driven Cardinality Estimation under Changing Workloads. PVLDB, 16(6): 1520 - 1533, 2023.
doi:10.14778/3583140.3583164

Proceedings of the VLDB Endowment, Vol. 16, No. 6 ISSN 2150-8097.
doi:10.14778/3583140.3583164

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/learnedsystems/CEB.

1 INTRODUCTION

Cardinality estimators are a critical part of query optimizers [23]. Traditional DBMSes' use simple cardinality estimators that optimize for practical concerns such as inference speed, predictability, and handling all types of filters — but this requires several simplifying assumptions (e.g., no correlation between columns or tables) which can lead to large estimation errors and suboptimal query plans. Several recent works [6, 9, 16, 18–20, 32, 38, 40, 41, 43–46] evaluate machine learning (ML) models for cardinality estimation — all these methods significantly improve estimation errors but have different practicality challenges.

ML-based cardinality estimators can broadly be divided into *data driven* and *query driven* methods. Conceptually, data driven methods model the joint distribution over all attributes in the database — e.g., using Deep Autoregressive Neural Networks [44, 45], or probabilistic graphical models [9, 16, 38, 43, 46] — which improve cardinality estimates compared to traditional methods by not relying on any simplifying assumptions. But accurately modeling the joint distribution of all attributes leads to large model sizes and slow inference times. Moreover, these methods typically do not support all kinds of queries: such as self joins or cyclic joins, or string LIKE filters.

Query driven models [6, 19, 20, 32, 40, 41], on the other hand, do not have these drawbacks. They learn regression models which map queries to their corresponding cardinalities using past workloads, without explicitly modeling the underlying data — therefore, these models can be lightweight and fast. Moreover, query driven models easily extend to all kinds of join patterns and filters. A key limitation of existing query driven methods, however, is that they do not generalize well to *workload drift*, i.e., new or changing workloads. For instance, slightly different query patterns (e.g., filters on new columns or tables), or data updates, can lead to unexpectedly bad query plans [32, 39], which makes these models impractical for real-world deployment.

To understand the failure cases of current query driven models, consider a purely query driven approach, in which the features only contain information about the SQL text. If there are filters on

a new table or column, then such models are doomed as they have no information to make useful estimates. A simple approach to improve generality is to augment query features with *data features*. For example, past work provides the database's estimate of the cardinality (based on a traditional estimator) as an input feature to the neural network [6, 32]. Since these estimates rely on up-to-date statistics (e.g., histograms) about all attributes, they provide useful information to the model even under workload drift. However, just providing DBMS estimates is not enough to ensure robustness. As we show, models trained using current techniques do not learn to utilize these features effectively. The reason is that the query features alone are often enough to perform very well on the training workload. Therefore, the model learns to rely mainly on these features during training, causing it to fail (sometimes catastrophically) under workload drift [10, 24, 25, 32, 39].

To avoid this problem, our idea is to emulate the characteristics of the workload drift scenarios for which we want robustness *during training*. Specifically, we partially mask query information during training, forcing the model to make predictions with missing information — this emulates scenarios with queries involving new tables or columns at test time. The DBMS estimate would still contain information from the masked part of the query, which the model uses together with the available query information to make predictions.

Our technique can be viewed as correcting an initial cardinality estimate, provided by the DBMS, using partial information about the query. Intuitively, the model learns the adjustments needed to improve upon the DBMS estimate. For example, if two columns are correlated, the DBMS estimate using the independence assumption would under-estimate the true cardinality; the model learns how much to inflate the estimate to account for the correlation. When there is a workload drift, the learned adjustment function would remain valid if the correlation structure has not changed significantly. Real-world data often exhibits consistent correlation structure over time even as the data changes (§3.3) provides several examples). In such cases, a learned "cardinality corrector" would continue to perform well. This reduces the need to retrain the model frequently, making it practical to deploy such models in dynamic settings.

A common way to provide the neural network information about the data correlations is through sampling. However, we find that current sampling featurization approaches cannot capture correlations across different tables in a join in the presence of workload drift. We develop a procedure based on sideways information passing [17] to provide sampling features which capture join correlations and are consistent and useful across workload drift scenarios. Finally, we show how to modify the training procedure, and these techniques, to train a model that adapts to data updates.

To evaluate robustness to query drift, we use three publicly released workloads on the Internet Movie Database (IMDb) with very different properties: Join Order Benchmark (JOB) [23], Cardinality Estimation Benchmark (CEB) [32], and JOBLight-train [19]. We also introduce a new workload on a public racing database, ErgastF1 [8] to verify our results on a different database. To evaluate robustness to data drift, we generate training cardinalities using these workloads with data only up to 1950 or 1980. We use one workload to train the model, and evaluate on the others. The key results on the final query performance are summarized in Figure 1, which

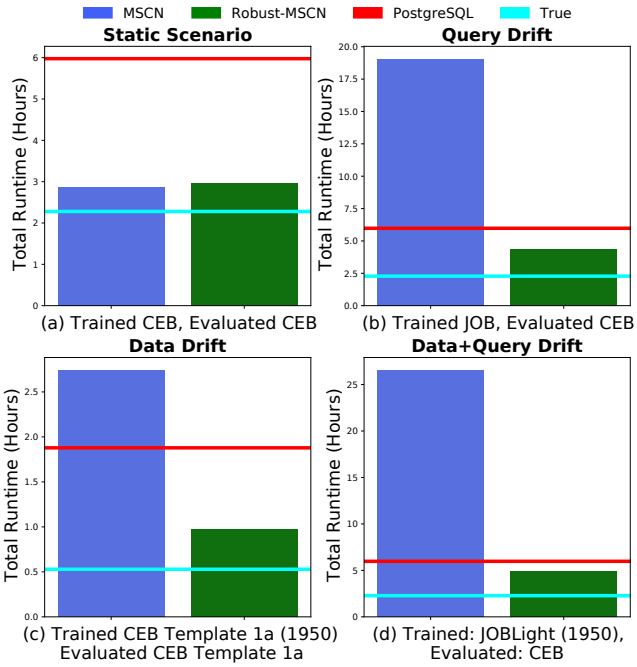


Figure 1: End to end query latencies of the MSCN model vs. our Robust-MSCN model in different workload drift scenarios. True cardinalities or PostgreSQL estimates are baselines.

includes a subset of our main experiments (§6). In a static scenario (no query/data drift), the state of the art MSCN model [19, 32] does very well. However, MSCN gets up to 5× slower than simply using PostgreSQL estimates in different workload drift scenarios. Meanwhile, the model trained with our techniques, Robust-MSCN, reliably improves query performance over PostgreSQL in each scenario — with speedups ranging from 1.2× to 2× despite workload drift.

Surprisingly, our model improves performance significantly over PostgreSQL on JOB, even when trained on the very simple JOBLight-train workload that only contains 40k synthetically generated queries with 3 tables and 2 joins each. Intuitively, the improvements occur when filters on just one or two tables have a dominant effect on the resulting cardinality. For instance, we find that for several large queries in JOB, involving dozen tables or more, most filters are redundant and the estimates of subplans are influenced by only a few filters for which similar patterns were seen in the simpler training workload (see §6.5 for details). This result shows that the learned model can effectively correct DBMS estimates using available query information, even outside of the training distribution. Therefore, performing well on such workloads does not require learning complicated joint distributions over all attributes, and effectively utilizing information even from a simple workload can be enough.

2 BACKGROUND

In this section, we will describe the details of query-driven models and their key benefits that motivate our extensive study.

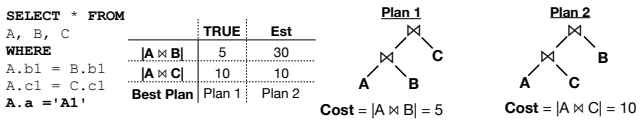


Figure 2: Simple example to describe evaluation functions.

2.1 Evaluation Functions

A query optimizer requires cardinality estimates for all the query's sub-plans, i.e., intermediate joins that it encounters when costing alternative query plans. For a simple example, consider Figure 2, which uses a simplified model costing each plan as the size of its intermediate result. In practice, each DBMS will have its own custom cost model, but the definition of the evaluation functions stays the same. The best plan, according to true estimates, is Plan 1 because $|A \bowtie B| < |A \bowtie C|$, however the estimator ('Est') overestimates $|A \bowtie B|$, therefore, it chooses sub-optimal Plan 2. This leads to a few ways to evaluate how good the estimates for a query, $\hat{y}$, are compared to the true values, y.

a) **Q-Error.** $max(\frac{y}{\hat{y}}, \frac{\hat{y}}{y})$. This is referred to as the Q-Error [29], or multiplicative error. In Figure 2, the average Q-Error of 'Est' would be $\frac{(30/5)+(10/10)}{2} = 3.5$.
b) **Query Runtime.** This is the execution time of the plan chosen using $\hat{y}$ — Plan 2 in Figure 2.
c) **DBMS Plan Cost.** This is a proxy for runtime using cost model units. Est's cardinalities lead to Plan 2, so, its cost is 10. Since cost units are hard to interpret, we will use the relative plan cost w.r.t the best plan. Here, Plan 1 costs 5, so we get $\frac{10}{5} = 2$ as the relative cost. This calculation depends on the cost model; if we use the PostgreSQL cost model, we will refer to it as the Relative PostgreSQL Plan Cost.

The ultimate goal is to optimize for query runtime. Even though Q-Error is a useful measure, recent works have shown the plan cost is better correlated to query runtime [11, 32]. Therefore, we use the DBMS plan cost metric when it is impractical to execute query plans, such as analyzing the performance of a model throughout its training phase.

2.2 Query Representation

In order to use a learned model (e.g. neural network), we need to represent, or *featurize*, the query in a compatible form.

Query Features. We refer to the tables, joins, and columns in the query as query features. We represent them using one-hot vectors. For e.g., if there are 10 tables in a workload, then each table will be represented by a length 10 vector with a 1 in the appropriate index. Typically, the training workload will contain several examples of the same tables, columns, or joins — thus, these one-hot vectors can be meaningfully interpreted by the model to learn correlations, or other patterns, from the workload data.

Data Features. We can run the DBMS estimator on a query, and use its output as a feature. This was proposed by Dutt et al. [7]. DBMS estimates rely on traditional cardinality estimators which are extremely fast due to various simplifying assumptions — thus, they do not add much to the inference time of the learned model, while providing the model with useful information.

Representing Filter Constants. It is harder to encode the filter predicate constants in a query because these have many more potential unique values. These constants can be for numerical range filters, categorical variables ('IN' or '=' clauses), or arbitrary strings in 'LIKE' clauses. However, it is also crucial for query driven methods as the cardinality depends on the effects of these filters. We will review the featurization approaches for categorical filters, which sets the backdrop for the techniques developed in §4. We can encode the filter constants *explicitly* (e.g., hashing [6, 32]) or *implicitly* (e.g., by filtering on a sample, and using *learned embeddings*.)

Explicit encoding. Explicit encoding is useful when constants repeat over workloads, possibly in different combinations. This approach does not help when there are filters with new constants, or on new columns — these would be ignored. An assumption made by models relying on explicitly encoding the filters is that the training workload is diverse enough to see most common constants, which is clearly not suitable for a workload drift scenario.

Sample bitmap. An example of implicit encoding are sample bitmaps, proposed by Kipf et al. [20]. They work as follows: for every table, keep a small sample (e.g., on IMDb, samples of size 1000 do well, with the actual size of the large tables ranging from $3M$ to $40M$). Execute the filter on the sample, and create a bitmap based on the output rows. Since particular constant values, such as genre = 'action' should map to particular rows in the sample, this approach can distinguish between different common constants. But notice that it is more general: it will produce a reasonable, and meaningful output even when there are filters on new columns, or LIKE filters. This utilizes the principle that there are many different ways to write a semantically similar SQL query, for instance, filters on different columns may actually have very similar effects when the columns are correlated. Therefore, it gives a signal for the correlation of attributes within a single table.

Learned embeddings. Another way to encode filters is to learn an embedding of the filter constants as a fixed-length vector. This has been done in databases using word2vec in Neo [28] or using contrastive learning [35]. Intuitively, filters that are correlated would be close to each other in the embedding space, even if they are on different columns. For instance, in Neo, every row tuple in the database is treated as a sentence, with each constant being a word. Then, Neo uses the unsupervised word2vec technique to map every filter constant to an embedding vector [28].

2.3 Why use query driven models?

The key challenge of query driven models for cardinality estimation is to be robust to queries that differ from the training workload. This is not a challenge for data driven models because they do not rely on the query workload. However, query-driven models have several advantages that make them an attractive choice. We now compare state-of-the-art query driven and data driven models in order to motivate our extensive study on the robustness of query driven models.

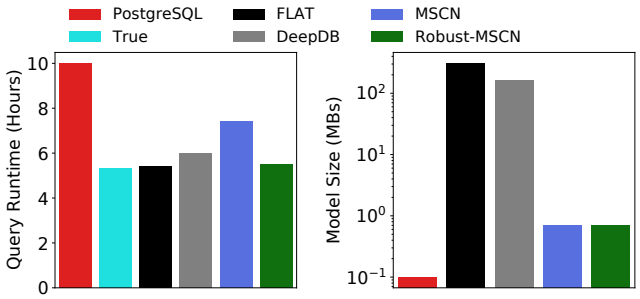


Figure 3: Comparing data-driven models with MSCN and Robust-MSCN.

Training / Evaluation workload. To directly compare the query driven and data driven models, we use the STATS-CEB benchmark [11] as the evaluation workload. Its key benefit is that the queries are supported by all data driven models. Figure 3 shows performance of several models on STATS-CEB in terms of end-to-end query latency and model sizes.

Runtime performance. The data driven models: FLAT [46] and DeepDB [16] are close to optimal when compared to the total query runtime of plans generated using true cardinalities. The query driven MSCN [19] model still improves over PostgreSQL, and our Robust-MSCN model is almost as good as the data driven models.

Model complexity. The data driven models achieve high accuracy by modeling the joint distribution of all attributes. This adds to model complexity, which is seen in the over 100× larger model sizes. This is despite STATS-CEB being a relatively small DB — with total size < 100 MB. This also causes similar slowdowns for training and inference time of data driven models.

If a query-driven model could reliably improve over PostgreSQL, even with workload drift, then their benefits: supporting all kinds of queries, low model complexity, faster training, and inference times, would make such models very attractive. However, current query-driven models can perform poorly under workload drift as we highlight next.

3 THE WORKLOAD DRIFT PROBLEM

There are many forms of workload drift which can have drastically different impacts on a learned model's performance. These include adding new data, or having new kinds of filters or filter constants, or filters on new columns and tables. A query-driven model learns patterns from a training workload, and applies them to new queries. We would not expect this approach to work well if no relevant pattern to the new queries was present in the training workload. But surprisingly, even in cases where relevant patterns do exist in the training workload, current query-driven models often fail under workload drift.

The root of the problem is that some of the patterns present in training are *spurious correlations* that do not generalize to new contexts. In this section, we will present simple microbenchmarks that provide examples of such spurious correlations. Although we leave the details of our techniques to §4, we provide insights on how biasing the model to learn to use more generalizable patterns during training can improve robustness to workload drift.

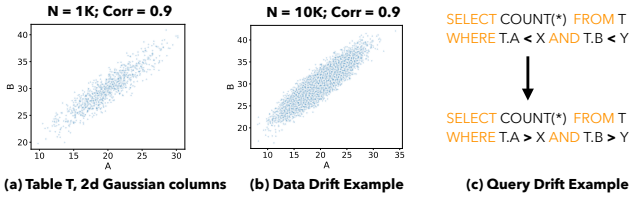
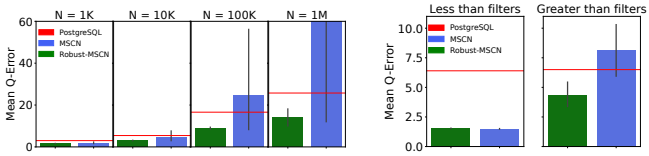


Figure 4: (a) For the query masking microbenchmarks we use a table, T, with correlated 2d gaussian columns, and parameters N (number of samples) and $Corr$ (correlation b/w the variables). (b),(c) show data and query drift examples.



(a) Varying N. **(b) Query drift.**

Figure 5: The performance of MSCN and Robust-MSCN model as we change size of the data, with same distribution (a), and as we move from '<' to '>' queries (b).

Setup. We test queries filtering a single table with two columns of correlated Gaussian variables (see Figure 4a for their distributions). We use two parameters to generate the table: the number of rows N, and the correlation between the two variables $corr$. The models are trained on several thousand queries from a particular data and workload distribution. Then, we evaluate them in the same setting or after the workload changes. We train each model five times and show standard deviation as error bars. For simplicity, we do not use sampling features for these examples.

3.1 Similar data insertion

In this experiment, we randomly sampled range filters on both columns A and B to generate the testing and training queries. We fix the correlation of the data to be 0.9.

Inserting more data (varying N). The models are trained on data from the table with $N = 1K$. Later, we insert tuples with $corr = 0.9$ into the table, increasing N to $10K$, $100K$, and $1M$ respectively, as shown in Figure 4b. From Figure 5a, we observe that the representative query-driven model MSCN does not adapt to data insertion. Specifically, MSCN is often significantly worse than the PostgreSQL baseline with a very large variance.

Note that as N increases, the only input feature that changes is the PostgreSQL estimate. Despite having this estimate as an input, MSCN does not use it effectively. Since $N = 1K$ throughout training, it is possible to produce good estimates using only the query filter features. Therefore, MSCN's learning procedure, which only optimizes for the loss (Q-Error) achieved on the training data, can result in many functions that appear to work equally well — sometimes, it may use the PostgreSQL estimate and sometimes it

may rely more on filter features. By contrast, our approach is to guide the learning procedure to emphasize the DBMS estimate. We encourage the model to learn to adjust the provided DBMS estimate using observed patterns in the training data. In this example, these patterns will be consistent as N increases because the DBMS's estimator (using the independence assumption) will consistently under-estimate the true cardinality in a similar way as observed during training.

3.2 Unseen features

We use the same experimental setup to study the effect of the query drift scenario shown in Figure 4c. Specifically, we generate the table with $N = 100K$ and $corr = 0.9$. Then, we generate training queries having only less than filters ('<') and consider a query drift with testing queries having only greater than filters ('>'). Since '>' does not appear in the training queries, the featurization will ignore the '>' operator. Thus, there will be some missing features when evaluating queries with the '>' operator. Other examples of such missing features could be a new table or column that was not seen in the training queries.

Figure 5b shows the results. Both the MSCN and Robust-MSCN models perform very well in the training template ('<' queries). However, the MSCN model has a significantly higher variance and a worse accuracy on the testing template ('>' queries). Although the Robust-MSCN model's performance also degrades outside the training template, it still manages to outperform the PostgreSQL baseline and has a much lower variance than the MSCN model. These results suggest a similar reasoning to the situation in §3.1. The original MSCN model can learn patterns that are not generalizable to queries beyond the training workload; for instance, in the less than queries, cardinalities are small when X and Y are small and increase proportionally as they increase. This relationship is reversed for '>' queries. The Robust-MSCN trained under our framework learns to correct the PostgreSQL estimate, which is an under-estimate for both '<' and '>' queries.

3.3 More complex workload drift

We provided two simple examples of workload drift, but the intuition underlying our method — that learning to correct DBMS estimates using patterns in training data results in more robust cardinality estimators — applies to many real-world scenarios. Real-world databases often exhibits consistent correlation patterns over time, causing traditional DBMS estimators to make consistent types of errors. Examples include: (1) Country and language are highly correlated in IMDb as movies get added over time. (2) Stock prices at the start and end of the day in a financial database. (3) Columns derived from other columns in a database, such as total and average sales, or date/time in different formats. In all these examples, the underlying correlation structure would persist through time regardless of data or query changes. Our evaluation explores several such real-world scenarios.

4 ROBUST TRAINING FRAMEWORK

4.1 Overview

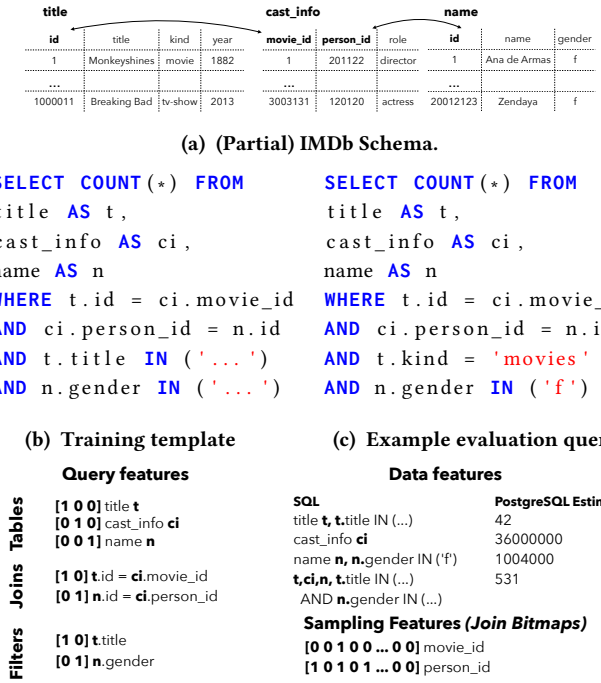
Our goal is to train a model to output cardinalities for a new query, and all of its subplans (i.e., queries involving some subset of the tables in the original queries). The model is trained using a query workload and its true cardinalities (training data). After training, the model is fixed, and only used for predicting cardinalities of new queries that are different from the training queries in some of the following aspects:

a) New queries have same templates, and only the filter constants change.
b) New queries include filters on new columns, new join graphs (new combination of tables), or entirely new tables.
c) New queries, and their cardinalities, are on updated data.

Several prior works have shown that different query driven models are very effective in the first scenario [6, 20, 32], but these models do not handle either of the second or third scenarios well [32, 39]. In the extreme case, these scenarios can be extended to evaluating the query driven models on a completely new database with a new workload — and naturally, we would not expect using a query driven model to be useful there. However, we seek to address the scenario where there is a smaller drift in the evaluation workload.

Current models can be very brittle, and lead to surprisingly bad query performance even with a relatively small shift in the workload. Such slight shifts are not uncommon: these could be exploratory queries that filter on different columns, a new user interested in slightly different questions, or just new data coming in over time. We seek to still do as well for parts of the query that have patterns observed in the training data, while effectively utilizing DBMS estimates to have reasonable accuracy for the rest.

An orthogonal approach to improve a query driven model in the presence of workload shift is to retrain it periodically. However,



(a) (Partial) IMDb Schema.

(b) Training template

(c) Example evaluation query

(d) Query representation for the example query.

Figure 6: Running example.

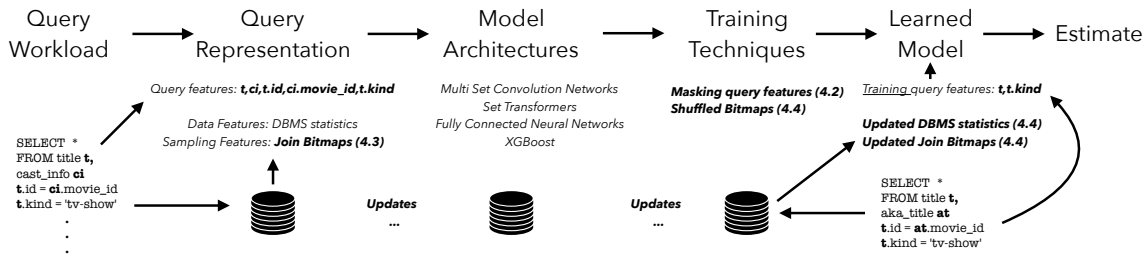


Figure 7: System overview. Our contributions are highlighted, with pointers to their section numbers.

retraining has its own associated cost — in particular, for collecting new training data — and our aim is to train models that are tolerant of some amount of workload drift, and therefore require less frequent retraining.

Running Example. To illustrate the key ideas, we will use simple pairs of training and evaluation workloads on IMDb shown in Figures 6b and 6c. We will assume that the model sees several examples with the form of a training template, and is then evaluated on a new query which differs from the training queries in some key aspect.

Figure 7 provides an overview of our training framework. We introduce techniques to mask query-related features during training, emphasizing DBMS estimates in the learned model and improving robustness to query workload drift (§4.2). In addition, we improve the query representation by developing join-bitmaps, a sampling-based feature that captures correlations across tables in a join (§4.3). During data updates, we provide the model up-to-date DBMS estimates and sampling features to further improve robustness (§4.4). We now describe each technique in turn.

4.2 Query Masking

In this section, we will introduce a new training scheme to make the model more robust to new query workloads with unseen columns or tables. The core idea is to stochastically hide some query-specific features during training and force the model to rely more on the data features, which are reliable across workload drift scenarios.

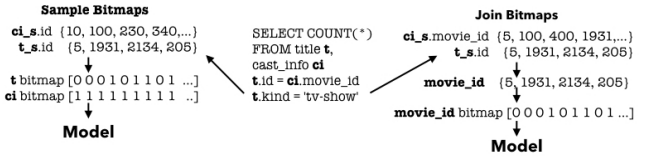
As an example, the training workload in Figure 6b contains filters selecting for specific movies. The evaluation queries in Figure 6c instead have a filter on a different column of 'title', i.e., 'title.kind = movies'. The cardinalities of these evaluation queries will be larger since the size of the 'title' table after applying the filter to the 'kind' attribute would be much larger than applying the filters to the 'title' attribute. The DBMS estimates for 'title' would reflect that the new queries should be much larger. But as discussed in §3, current techniques could learn to map all queries containing 'title' to small values.

Masking technique. We randomly zero out each query feature with probability p *during training*. This is not the same as simply not having query features, as most query features would still be present. Instead, this is akin to forcing the model to predict cardinalities with missing query information by relying more on the data features. At test time, when the model is used to estimate new queries, we will provide all *available* query features — if the query involves a new column or table, this will just be represented as zeros.

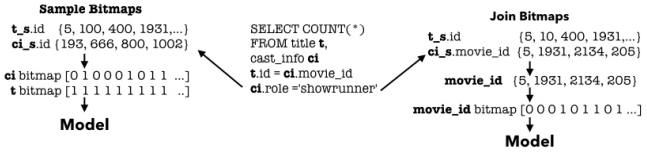
This is similar to the common ML technique called dropout, although the motivation is different here. Dropout is typically used on all input features or on the hidden layers, and has been shown to be useful for regularization [3]. However, in tasks such as image processing, the models often serve as feature extractors and do not apply dropout selectively to specific elements of the input. Nevertheless, the success of dropout shows that it is not unreasonable to ask a model to make predictions with missing information.

Alternatives. We experimented with other ways to try to achieve similar benefits. For instance, changing the target label of the ML models from the true cardinalities to the difference between the DBMS estimate and the true cardinality also increases the reliance of the model on the DBMS estimate. In some scenarios, like data updates, this was equally helpful, but it did not help as much with new tables or columns. The query masking technique is a more effective approach because it combines two benefits: (i) increased reliance on data features, and (ii) robustness to missing information. Regularization techniques could in theory help the model learn simpler functions that increase reliance on the DBMS estimate, but we found general-purpose regularizers such as adding an $L1/L2$ penalty on model's weights did not help improve robustness to workload drift scenarios. A complimentary approach to improve robustness is to modify the loss function to promote simpler models, as in Flow-Loss [32]. Query masking is more widely applicable and can be combined with any loss function (see §7 for a more detailed discussion).

4.3 Join Bitmaps



(a) Example training query.



(b) Example evaluation query.

Figure 8: Comparing sample bitmap vs. join bitmap.

Join bitmap. Figure 8a shows how sample bitmap and join bitmap differ in the context of a simple two table join. The sample bitmap approach will have two independent samples on 'title' and 'cast_info'. The filter on title selects for 'tv-shows' — this filter is applied on title's sample table, and only a few rows are selected. Since no filter is applied on 'cast_info', all rows from the sample are selected. These values are then hashed to bit-vectors, which are part of the query representation processed by the model. The join bitmap approach creates a sample on 'title.id' (the primary key), and a correlated sample over the same values on 'cast_info.movie_id' (the foreign key). Then, the filters on title and cast_info are applied to the correlated samples. The join condition, 'title.id = cast_info.movie_id' is enforced by the intersection of the results on these samples — which finally gives us the single bitvector to be processed by the model. Thus, there will be one bitmap per primary key in the query representation, with the information from the foreign key samples included in this bitmap. Figure 8b shows an example where a correlated filter to Figure 8a is used, but on the 'cast_info' table. The join bitmap approach will continue to model the effect of the filter on the 'movie_id' key — which will give similar featurization to the training query in Figure 8a.

Join sampling efficiency. A classical result in the sampling literature states that if you have a uniform and independent sample of two tables, then the quality of a sample drops quadratically for their join; i.e., if we started with two 1% samples, their join would reflect a 0.1% sample from the joined table [2]. This is because there might be many misses in which the join key value selected in one sample is not present in another. However, the approach we proposed uses correlated sampling — e.g., after we have selected 1000 values from 'title.id', we create samples with exactly those values in other tables where 'title.id' is a foreign key.

Alternatives. Another approach to capture correlations across tables that we considered was using learned embeddings [28, 35]. This works quite well on the example in Figure 8 because it would generate similar embeddings for the filters 't.kind = tv-show' and 'ci.role=showrunner'. However, this approach cannot be extended to support cases with multiple filter constants, e.g., 'IN' queries. Specifically, it will provide several embedding vectors for the 'IN' filter and average them, which will not be meaningful in the embedding space. Alternatively, our sampling-based approach can directly apply the 'IN' filter (or any other type of filter) on the sample, and the resulting bitmap would consistently represent the semantics of the filter.

4.4 Data Updates

Data updates can mean that the ground truth cardinalities used in the training workload may now be wrong — thus, models learned to predict those cardinalities would degrade as well. Existing query driven models inevitably recommend collecting new training data (or updating old training queries) by executing the queries, and retraining their models [24–26]. As these models need a large number of queries to retrain, collecting the new training data is the main bottleneck in retraining the model. Therefore, existing query driven methods are generally believed to be unsuitable for dynamic databases, where data updates frequently [11, 43].

Our key insight is that information about data updates can be encoded in the data and sampling features. However, we need to ensure that the learned model actually understands and utilizes this information. For instance, we would like the model to learn that filters on two columns are often correlated, but the DBMS estimate may be underestimating it because of the independence assumption, so the model could learn to correct the DBMS estimate upward. As the data updates, the exact estimate may be much larger, but the correction pattern of DBMS under-estimation would still hold in general. For a concrete example, let us assume that we trained our model on data from the 1950 version of IMDb. Then, consider the following query:

```
SELECT COUNT(*)
FROM cast_info AS ci, name AS n
WHERE ci.person_id = n.id AND n.gender = 'f'
AND ci.role = 'actress'
```

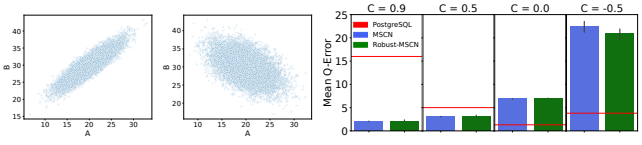
In the full IMDb version, the cardinalities would be much higher than the 1950 version. However, the strong correlation between 'n.gender=f' and 'ci.role=actress' will be preserved.

Thus, query driven models trained within our framework can be made robust against data update to a much larger extent. We expect the quality of the trained model to drop as data updates, but the drop is gradual and avoids unpredictably bad performance. Thus, even in a dynamic scenario, we can continue to get the benefits of our trained model, and will require retraining the model much less frequently. This can be useful even with highly dynamic databases, where a model retraining may be scheduled periodically, with our training techniques allowing us to trust the learned model in periods between the retraining.

Updating data features. The data features used in our frameworks are derived from DBMS estimators (e.g. histograms). These estimators generally use simplified assumptions to maintain per-attribute statistics for cardinality estimation. Therefore, they can easily keep up with fast data updates and our framework can use these updated DBMS estimates as new data features dynamically. In PostgreSQL, for instance, the 'VACUUM ANALYZE' command should update all these statistics in just a few minutes on IMDb.

Sampling join bitmap on-the-fly. It is trickier to update sampling features than data features. Existing query driven cardinality estimators keep a fixed data sample on which filters are applied to generate bitmaps. The trained model will only learn the correlations captured by this fixed sample. As data updates, the original sample would no longer be a uniform sample; for e.g., queries that select new data will always have zero hits. Similar to data features, we could periodically create new updated samples in order to keep up with data updates, or even sample join bitmaps on-the-fly using techniques such as reservoir sampling. This will ensure the samples are fresh at inference time.

At training time, we emulate these bitmaps by *shuffling* the bitmap indices at each step. At first, this may seem to lose the benefits of the bitmaps. The indices in the bitmap will no longer be meaningful. However, the bitmap will still carry a useful signal — capturing the join correlations, and distinguishing between broad patterns, such as a lot of rows being selected, very few rows being selected, and so on. Compared to using a fixed sample, shuffling



(a) Corr=0.9. **(b) Corr=-0.5.** **(c) Results.**

Figure 9: Data drift for the correlated gaussians. (a) presents the training data, $corr = 0.9$.(b) shows a drift, $corr = -0.5$.

bitmaps will lose performance as the learned model will not be able to learn correlations that relied on the specific indices in the sample. Therefore, in the non-update scenario, we use the models with static bitmaps. In general, since data updates are easily noticeable, it is possible to choose a model based on whether it requires robustness to data updates or not.

4.5 Limitations

In this section, we will discuss the drawbacks of the query masking and join bitmaps techniques discussed above.

4.5.1 Query Masking.

The effect of query masking is to increase the reliance of the models on the DBMS estimates. DBMS estimators make strong uniformity and independence assumptions about the data. A model trained with masking would learn to correct for the errors caused by these assumptions, and we would expect it to generalize better to new workloads where the DBMS estimator makes similar kinds of errors, as shown in the microbenchmarks in §3.1.

However, if the nature of the errors in the DBMS estimates change — e.g., because the data correlations are inherently different in the training and evaluation workloads — then, this approach would not help. As an example, consider the case of correlated 2d gaussian variables again. We will keep the number of samples, N, fixed and vary the correlation between the two variables. An example is shown in Figures 9a and 9b. The models are trained with samples from $corr = 0.9$. The results are shown in Figure 9c.

Both the MSCN and Robust-MSCN models perform worse as correlation changes, getting progressively worse than PostgreSQL estimates as the correlation deviates further from the training regime. Robust-MSCN does not help in this scenario because it had learned to correct for PostgreSQL's under-estimates due to the independence assumption. At $corr = 0.5$, the models still improve a little over PostgreSQL as the variables are still correlated like the training regime. When $corr = 0.0$, PostgreSQL estimates are much more precise and do not require corrections, but our learned model continues to treat them as under-estimates from the $corr = 0.9$ regime observed during training. When $corr = -0.5$, the PostgreSQL would over-estimate, but the model would now "correct" it in the wrong direction.

4.5.2 Join Bitmaps.

Sampling overhead. Join bitmaps adds additional overhead at inference time compared to the standard sample bitmap in two ways: (1) The sample sizes are slightly larger since the multiplicity of each value in a primary table is larger when it is a foreign key. We limit this by having a maximum number of entries in the samples on foreign keys. (2) The number of joined columns is more than the number of tables, i.e., more samples are needed. At inference time, sample bitmap will execute the filters on each table's sample. Join bitmaps executes these filters on each join key sample.

Only supports equi-joins. We efficiently combine samples on all equivalent join keys by taking their intersection — this works only for equi-joins. For arbitrary join types, one would essentially need to perform the join on the sample which is more expensive.

Beyond primary - foreign key joins. We require existing sample tables on the join key before inference. This is only practical for known join keys, such as primary / foreign keys, and join bitmaps could not be created on ad-hoc join queries on new columns.

Correlations involving multiple primary keys. There is no easy way to efficiently capture the correlations across equi-joins involving more than one primary key in a single bitmap. For instance, consider a query that filters for movies of Robert Downey Jr, which will need a filter on the table 'names'. This also filters out most titles, but it will not be captured in the join bitmap of 'title.id'. The join bitmap on the primary key 'name.id' should capture it. This suggests the requirement for training workloads: we would like to see coverage of bitmaps on all the primary keys involved in the evaluation.

5 MODELS AND WORKLOADS

In this section we go over the models and workloads used in the experiments, and the trade-offs with other alternatives.

5.1 Query driven models

First, we describe how the features are used by different neural network architectures, and design decisions we make with a focus on robustness to workload drift. There are two class of architectures that have been proposed for query driven models, which differ in how the inputs are represented.

1d featurization. Flattens all the query information into a 1d vector, and then uses standard learning methods, such as XGboost [6], or fully connected neural networks [7, 31, 40]. The idea is to assign an index to every table, join key, or column in the vector, and put the representation of it at that index. Its benefits include simplicity, and interpretability for tree based methods such as XGBoost. It has two problems: (1) Input sizes can get large, e.g., when using bitmaps, since all bitmaps would be concatenated. (2) It doesn't represent information from outside training, e.g., new columns or joins.

Set featurization. In set based architectures, the set of tables, joins, and filters in a query are considered independently. Each table, join, or filter is featurized to a fixed length 1d vector. An example is Multi Set Convolutional Networks (MSCN) [13, 19, 20, 32]. It processes the three sets of fixed length vectors for tables, joins, or filters independently using a fully connected network, and output a single vector by averaging for each. These vectors are concatenated together and processed further by fully connected layers. These set features are more suited for workload drift, such as self-joins or having new table / joins / columns. A new table will not have an appropriate query features, as discussed in §4.2, however, we can still embed information about it using the DBMS estimate or bitmaps. Therefore, in our evaluation, we focus on set architectures.

Adding data features to set networks. The original MSCN architecture [19] did not use DBMS data features. We append the data feature to each hidden layer after the sets are concatenated. This further enhances the importance of the DBMS estimate, which can be relied upon in workload drift scenarios.

5.2 Workloads

Here we describe the key properties of the workloads used, and how we use them to test challenging workload drift scenarios. Several different workloads have been released on the Internet Movie Database (IMDb) over the years. We select three such workloads with very different properties, but over the same schema and database. We utilize this to train our models on one workload, and test on other workloads to rigorously evaluate workload drift.

Join Order Benchmark (JOB). JOB [23] is a set of 113 handwritten queries with up to 16 joins. This does not contain a separate training workload. However, several papers have used it as a test workload for query runtimes [4, 5, 14, 27, 28, 32].

Cardinality Estimation Benchmark (CEB). CEB [32] uses handcrafted templates, and query generation rules to create challenging large queries, like JOB, but with several thousand queries — thus, having both training and test workloads. In comparison, JOB contains 5 additional tables, and several more columns.

JOBLight-train. JOBLight-train [20] contains synthetically generated 40K queries with 3-table joins. Since there are only two joins per query, the workload is trivial as a test workload for query optimization. However, it is a useful training workload since it is very simple, and contains several differences with the other workloads.

Evaluating data updates. The full IMDb database we use contains data up to 2013 [32]. We create two additional versions of IMDb: IMDb-1950, and IMDb-1980, in which we filter out all movies after 1950 and 1980 (and related entries in other tables). IMDb-1950 has less than 10% of the full IMDb data, and IMDb-1980 has about 30%. These old versions of the database are just used for training. We regenerate the ground truth cardinalities of all the queries in JOBLight-train, and one of the templates in CEB (since generating ground truth data on CEB templates takes much longer — requiring almost a month of execution for the full workload). We use these as training workloads, and evaluate the models on the full IMDb.

ErgastF1. We introduce a new challenging cardinality estimation workload, ErgastF1. The workload is on the ErgastF1 database, as used in [15]. It uses the templating scheme from [32] to construct queries with several joins (up to 8) which induce large errors from PostgreSQL. We also construct a 'Simple-ErgastF1' workload, which follows the design of JOBLight-train, including only up to 3 tables, and random filters. 'Simple-ErgastF1' is used to train the models, in order to test workload drift on 'ErgastF1' workload.

6 EXPERIMENTS

6.1 Setup

Baselines. As baselines to compare the learned models against, we use: (1) True cardinalities. This represents the best query runtime performance we can achieve. (2) PostgreSQL estimates (3) Microsoft SQL-Server estimates. PostgreSQL estimates are also provided as data features to the models. SQL-Server represents the best traditional estimator using more sophisticated strategies than PostgreSQL [1].

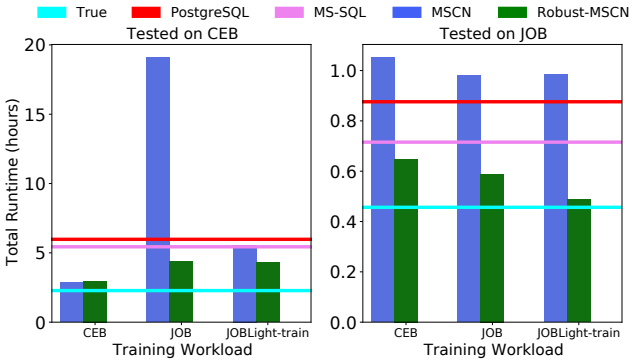


Figure 10: Runtime performance on PostgreSQL of baselines, and models trained on CEB, JOB, or JOBLight-train.

Learned Models. We will be comparing the MSCN and Robust-MSCN models. The MSCN model is trained as in past works with bitmaps of size 1000 [19]. Robust-MSCN model uses query masking with $p = 0.2$, primary key bitmap size of 1000, and a foreign key bitmap size of at most 1% of the table (which is at most 30K on IMDb). In the online appendix [30], we show that the results presented here are not very sensitive to these hyperparameter choices. We use Q-Error as the loss function to train all the models.

Execution environment. We use the execution environment provided with CEB [33], using a docker container with PostgreSQL 12, and 1GB RAM. We clear cache before every execution, and use primary and foreign key indexes.

Learning curves. In a few scenarios, to better understand the learned models, we plot error metrics after each epoch of a model's training, i.e., after each pass over the full training data.

Training / Test workload splits. We train our models on JOBLight-train, JOB, or CEB. We use JOB and CEB as the test workloads as they have complex queries where better cardinality estimates clearly improve query plans. When testing on CEB, we use a test set of 509 queries [33]. When training on CEB, we use a training set of 2600 queries [33]. When testing on JOB, we use all 113 queries. Because JOB has much fewer queries, when training and testing on it, we use ten different 50 − 50 training/test splits. This scenario is an example of a workload drift scenario as well because each JOB query has a slightly different template.

6.2 Cross Workload Generalization

In Figure 10, we show the end to end runtime of query plans on CEB and JOB generated from estimates of the MSCN and Robust-MSCN model trained on different workloads. Robust-MSCN model shows non-trivial improvements over PostgreSQL across all scenarios.

Evaluating on new queries from same workload. When trained and tested on CEB, both the models do clearly better than PostgreSQL, and are close to optimal.

MSCN is brittle when tested on different workloads. It is expected that workload drift should make a learned model perform

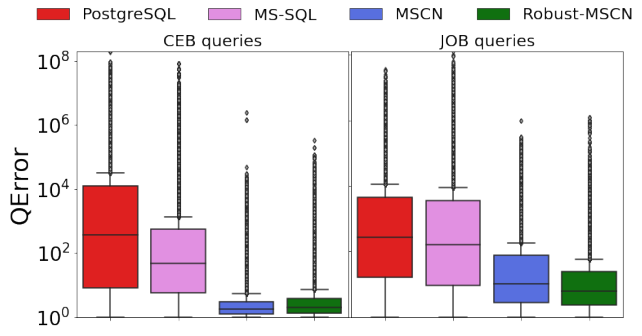


Figure 11: Q-Error of baselines, and models trained on CEB, and tested on CEB (left) or JOB (right).

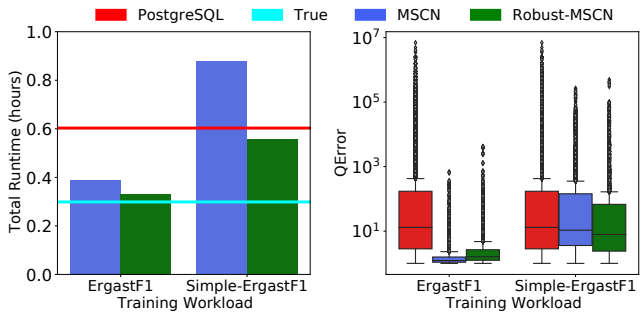


Figure 12: Results on ErgastF1 workload for models trained on it, or trained on a simple workload with only two joins.

worse, however, it is problematic when models may output noisy or inconsistent estimates for parts of a query that are outside the training regime. This may not always lead to worse query plans; It is reflected in the MSCN model' total latencies getting worse by different amounts, with worse total runtime than PostgreSQL in three of the scenarios. Meanwhile Robust-MSCN improves over PostgreSQL in each scenario.

Q-Errors. The distribution of Q-Errors for models trained on CEB, and tested on CEB or JOB — is shown in Figure 11. On the same workload, both models improve drastically over PostgreSQL. On the new workload, both models again improve over PostgreSQL, with the Robust-MSCN model having lower median and 90th percentile Q-Error than MSCN. However, despite improving Q-Error compared to PostgreSQL, as we saw in Figure 10, MSCN has worse runtime. This discrepancy makes sense when you consider that Q-Errors on JOB are over $70K$ subplan estimates — and only improving overall Q-Error estimates is not enough to guarantee better end to end performance in the 113 query plans.

New database. We find similar patterns in performance also show on a different database. Figure 12 shows the runtime performance and Q-Errors on ErgastF1. When the training and test workload are from the same distribution, both models clearly improve over PostgreSQL and are close to optimal. However, on the simple workload, MSCN's query performance gets worse than PostgreSQL while Robust-MSCN degrades much less.

6.3 Data Updates

For the data updates experiments, the Robust-MSCN model will include the updated join bitmaps and shuffled bitmaps training approach described in §4.4.

Only data drift. Figure 14 shows the results of models trained on CEB template 1*a* with ground truth data from IMDb-1950, and tested on unseen queries from the same template. For comparison, we also show Robust-MSCN trained on the full IMDb — this would be the static scenario without any workload drift, and both Robust-MSCN and MSCN models do equally well here. In terms of total runtime, the Robust-MSCN model loses only a little performance despite the old data, while the MSCN model gets much worse than PostgreSQL when trained on 1950 data. The learning curves for Relative PostgreSQL Cost and Q-Error in Figure 14 include error bars over three runs for each model. The Robust-MSCN model trained on 1950 data has more variance, but over time it converges close to the performance of the Robust-MSCN model trained with the latest data.

Workload drift + data drift. Figure 15 shows the result of training MSCN or Robust-MSCN on the JOBLight-train workload from IMDb-1950, and IMDb-1980, and evaluating on the full IMDb version. In general, we don't see the consistent improvements over PostgreSQL estimates in terms of query latency as when we trained on IMDb-full itself — which suggests scope for developing better techniques in such scenarios. However, Robust-MSCN is better than if we had just used MSCN on the same training data — where the performance drops up to 4 – 5× when trained on stale data. This is one of the benefits: even in highly challenging scenarios, the Robust-MSCN approach doesn't get unpredictably bad.

6.4 Ablation Study

In this section, we tease apart the individual effect of our proposed techniques, and the different kind of features, by an ablation study. Figure 16 shows the results of training a model on JOBLight-train after some key modifications compared to the Robust-MSCN model.

No query features. Throughout our scheme, we have focused a lot on effectively utilizing the data and sampling features. It is natural to ask if the query features are even required? As we see, there is a noticeable degradation in performance without the query features — especially on Q-Errors. This is not surprising: the key benefit of query driven models is that it makes it possible to learn correlations between certain attributes / joins or tables in a compact form — removing all query information makes this impossible.

Effect of masking. Consider the 'Mask All Features' ablation. In this, we apply masking/dropout (§4.2) to the data features as well. This evaluates if the benefits seen were just due to the regularization properties of dropout, or because of the DBMS specific adaptation of the idea we use here. As we see, the 'Mask All Features' performs similar to 'No Query Masking' on JOB, and both of these get worse than PostgreSQL when tested on CEB..

Overlap between sampling and data features. Both the sampling and data features capture information about the underlying data — interestingly, the model still does quite well on JOB if we just remove one of them. However, the performance degrades drastically when both are removed. On CEB, the performance is particularly

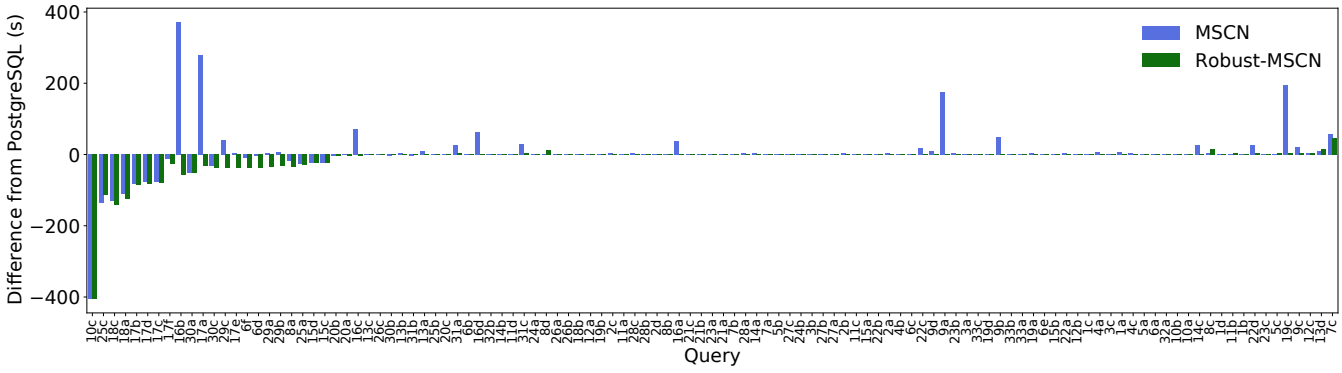


Figure 13: Per query results for models trained on JOBLight-train and evaluated on JOB. (Lower is better).

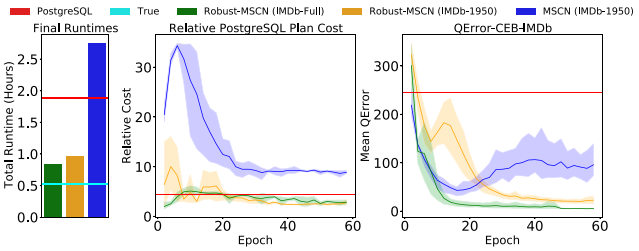


Figure 14: Final runtimes, and learning curves for models evaluated on the data drift scenario.

Figure 15: Models trained on old IMDb versions and JOBLight-train workload. Evaluation is on full IMDb.

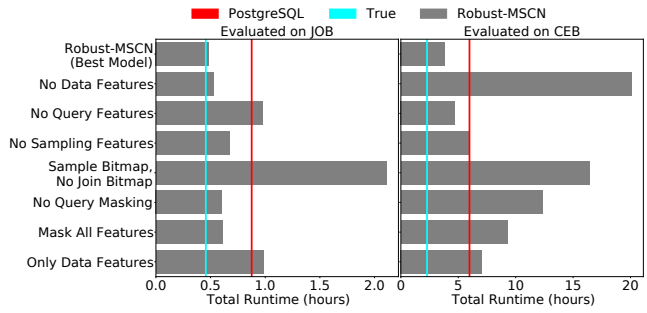
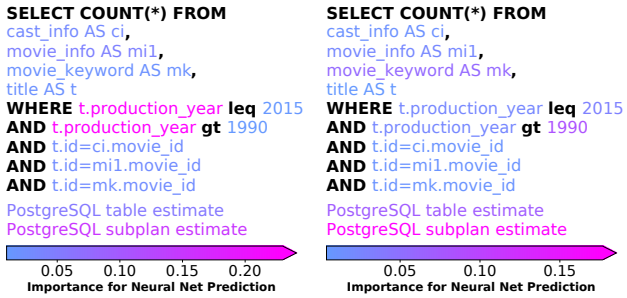


Figure 16: Ablation study. Each label (y-axis) is a difference from the Robust-MSCN model trained on JOBLight-train.



(a) MSCN's importance. **(b) Robust-MSCN's importance.**

Figure 17: Feature importance of MSCN vs Robust-MSCN.

bad if the data features are removed. We believe that this is because of the extreme nature of filters on JOB — often, having a coarse signal, such as the output being present / or not present in the bitmap, is enough to learn a reasonable model there.

6.5 Understanding JOBLight-train performance

Since JOBLight-train is such a simple workload, the improvements of Robust-MSCN in terms of total runtime are particularly surprising. We will analyze these results in greater detail — although the themes we highlight are also applicable to the other scenarios shown above.

Where do the MSCN and Robust-MSCN model performances differ? Figure 13 shows both the model's runtime difference from PostgreSQL on all JOB queries. Both models are able to correct PostgreSQL when its significantly wrong — there are several queries where they improve on PostgreSQL by over 100 seconds. However, the MSCN model cancels these out with several regressions of similar magnitude — showing its brittleness to workload drift. Robust-MSCN has much fewer, and smaller, regressions.

Interpreting learned models. We apply interpretablity techniques developed to understand what a deep neural network learns

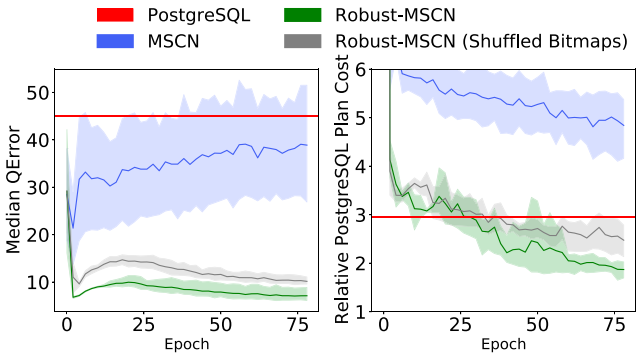


Figure 18: JOB Learning curves, trained on JOBLight-train.

to the MSCN and Robust-MSCN model trained on JOBLight-train. We use the integrated gradients algorithm [37] implemented in the Captum library [21]. Figure 17 shows the relative importance of each input in a CEB SQL query to the models. This gives us insight into what the two models learned — and they are quite different despite having the same training workload. For Robust-MSCN, the importance for the PostgreSQL estimate is the highest. This is in line with our intuition behind the query masking technique (§4.2). MSCN relies more on the query features, such as the presence of the column 't.production_year', which is less robust for workload drifts. For instance, the 'production_year' feature will be very misleading if the model was trained with IMDb-1950 data, and tested on the latest data.

Learning curves. Consider the learning curves of the MSCN and Robust-MSCN model trained on JOBLight-train (Figure 18). In the first couple of epochs, both models go from random estimates, to reasonable estimates that improve accuracy on the very different JOB queries. However, beyond that, the MSCN model continues to overfit to its training workload (JOBLight-train), and progressively gets worse on JOB. The Robust-MSCN model continues to improve Q-Error on JOB throughout training, therefore, it shows that it is learning generalizable and useful features rather than overfitting. Intuitively, it suggests that it learns to correct cardinalities wherever there is a strong signal. Moreover, these improvements translate to consistent gains in the plan costs as well — which suggests that the model provides consistent estimates for all subplans in a query.

Comparison with shuffled bitmaps. Recall that in the static scenario, we did not use the shuffled bitmap approach from §4.4. In the learning curves in Figure 18, we show the performance penalty we would have if we used the shuffled bitmap approach when there were no data updates.

7 RELATED WORK

Many early works using ML in DBMS were for improving cardinality estimation [9, 22, 36], and several classical approaches have been proposed for it, but it has remained a challenging and open problem. Recent ML research for cardinality estimation considers two main approaches.

Data driven models. Data driven approaches draw from the wide ML literature on modeling joint probability distributions using deep autoregressive models [44], sum product networks [16, 46], or probabilistic graphical models [38, 42, 43], or other methods [12, 16, 34, 41, 45].

Query driven models. Query driven approaches generate ground truth cardinalities for a given workload, and use a regression model to map the queries to their cardinalities. This includes MSCN [19], lightweight models [6, 7], and several other related variants [6, 13, 14, 20, 32, 40, 41].

Workload drift. Several works have noted the workload drift challenge for query driven cardinality estimation models [10, 24, 25, 32, 39]. We expand on this by exploring the causes that lead to workload drift, and propose a solution framework for it.

Flow-Loss. An alternative loss function to Q-Error, proposed in [32], is Flow-Loss, which is a differentiable approximation of the plan cost. This requires it to model the end to end query optimization process using approximations and differentiable primitives — which require several assumptions. If the differentiable cost model is a reasonable approximation, then this loss function leads to more robust models, as shown in [32]. The authors showed that models trained with Flow-Loss generalize better to new unseen templates. This is an example of query workload drift. In this paper, we considered more extensive drifts. Our approach is complementary to Flow-Loss, and it can be used to improve Flow-Loss trained models further. But, more importantly, our approach is much simpler, easier to interpret and implement, and can make the models optimized using Q-Error more robust than ones just using Flow-Loss. However, when the training workload contains very simple queries, like JOBLight-train— Flow-Loss models may not learn to predict accurate cardinalities for complex queries (see online appendix [30]).

Retraining models. Query driven approaches assume retraining models in case of new workloads. Warper [24] builds on this idea: it automatically detects workload drift, generates similar queries, collects ground truth, and retrains the model. This approach is complementary to the techniques described in our work — we will require retraining models as well, but much less frequently.

8 CONCLUSION

We show that appropriately trained query driven models can remain very useful in a much wider settings than previously considered. Previously, a query driven model would be considered obsolete as query patterns changed, or data updates [24]. This would require retraining, and potentially a very expensive data collection process. Thus, it is not ideal to do it every time workload drift occurs.Instead, with our techniques, we could trust our model to adapt gracefully to workload drift, utilizing its training workload to improve where it can, and being reasonably anchored to the baseline DBMS performance in cases where it is not possible to improve. And as query or data patterns keep changing, retraining could be triggered periodically.

ACKNOWLEDGMENTS

This research was supported by Intel as part of the MIT Data Systems and AI Lab (DSAIL), and partially supported by NSF award numbers 1900933 and 1751009.

REFERENCES

[1] 2018. SQL Server's Join Cardinality Estimation. https://www.sqlshack.com/join-estimation-internals/ [Online;].

[2] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. The Aqua Approximate Query Answering System. In SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA, Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh (Eds.). ACM Press, 574–576. https://doi.org/10.1145/304182.304581

[3] Pierre Baldi and Peter J Sadowski. 2013. Understanding dropout. Advances in neural information processing systems 26 (2013).

[4] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities. In Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 18–35. https://doi.org/10.1145/3299869.3319894

[5] Asoke Datta, Yesdaulet Izenov, Brian Tsan, and Florin Rusu. 2021. Simpli-Squared: A Very Simple Yet Unexpectedly Powerful Join Ordering Algorithm Without Cardinality Estimates. arXiv preprint arXiv:2111.00163 (2021).

[6] Anshuman Dutt, Chi Wang, Vivek R. Narasayya, and Surajit Chaudhuri. 2020. Efficiently Approximating Selectivity Functions using Low Overhead Regression Models. Proc. VLDB Endow. 13, 11 (2020), 2215–2228. http://www.vldb.org/pvldb/vol13/p2215-dutt.pdf

[7] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates using Lightweight Models. Proc. VLDB Endow. 12, 9 (2019), 1044–1057. https://doi.org/10.14778/3329772.3329780

[8] Ergast. 2021. Ergast F1 Database Schema. https://relational.fit.cvut.cz/assets/img/datasets-generated/ErgastF1.svg [Online;].

[9] Lise Getoor, Benjamin Taskar, and Daphne Koller. 2001. Selectivity Estimation using Probabilistic Models. In Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001, Sharad Mehrotra and Timos K. Sellis (Eds.). ACM, 461–472. https://doi.org/10.1145/375663.375727

[10] Max Halford, Philippe Saint-Pierre, and Franck Morvan. 2020. Selectivity correction with online machine learning. arXiv preprint arXiv:2009.09884 (2020).

[11] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. Proc. VLDB Endow. 15, 4 (2021), 752–765. https://doi.org/10.14778/3503585.3503586

[12] 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 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1035–1050. https://doi.org/10.1145/3318464.3389741

[13] Rojeh Hayek and Oded Shmueli. 2020. Improved Cardinality Estimation by Learning Queries Containment Rates. In Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020, Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang (Eds.). OpenProceedings.org, 157–168. https://doi.org/10.5441/002/edbt.2020.15

[14] Axel Hertzschuch, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. 2021. Simplicity Done Right for Join Ordering. In 11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings. www.cidrdb.org. http://cidrdb.org/cidr2021/papers/cidr2021_paper01.pdf

[15] Benjamin Hilprecht and Carsten Binnig. 2021. One model to rule them all: towards zero-shot learning for databases. arXiv preprint arXiv:2105.00642 (2021).

[16] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! Proc. VLDB Endow. 13, 7 (2020), 992–1005. https://doi.org/10.14778/3384345.3384349

[17] Zachary G Ives and Nicholas E Taylor. 2008. Sideways information passing for push-style query processing. (2008).

[18] Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. 2022. Learned cardinality estimation: An in-depth study. In Proceedings of the 2022 International Conference on Management of Data. 1214–1227.

[19] Andreas Kipf, Michael Freitag, Dimitri Vorona, Peter Boncz, Thomas Neumann, and Alfons Kemper. 2019. Estimating filtered group-by queries is hard: Deep learning to the rescue. In 1st International Workshop on Applied AI for Database Systems and Applications.

[20] Andreas Kipf, Dimitri Vorona, Jonas Müller, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, Thomas Neumann, and Alfons Kemper. 2019. Estimating Cardinalities with Deep Sketches. In Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1937–1940. https://doi.org/10.1145/3299869.3320218

[21] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, et al. 2020. Captum: A unified and generic model interpretability library for pytorch. arXiv preprint arXiv:2009.07896 (2020).

[22] M. Seetha Lakshmi and Shaoyu Zhou. 1998. Selectivity Estimation in Extensible Databases - A Neural Network Approach. In VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA, Ashish Gupta, Oded Shmueli, and Jennifer Widom (Eds.). Morgan Kaufmann, 623–627. http://www.vldb.org/conf/1998/p623.pdf

[23] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? Proc. VLDB Endow. 9, 3 (2015), 204–215. https://doi.org/10.14778/2850583.2850594

[24] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts. In Proceedings of the 2022 International Conference on Management of Data.

[25] Beibin Li, Yao Lu, Chi Wang, and Srikanth Kandula. 2021. Cardinality Estimation: Is Machine Learning a Silver Bullet. In 3rd International Workshop on Applied AI for Database Systems and Applications (AIDB).

[26] Jie Liu, Wenqian Dong, Qingqing Zhou, and Dong Li. 2021. Fauce: fast and accurate deep ensembles with uncertainty for cardinality estimation. Proceedings of the VLDB Endowment 14, 11 (2021), 1950–1963.

[27] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2022. Bao: Making learned query optimization practical. ACM SIGMOD Record 51, 1 (2022), 6–13.

[28] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A learned query optimizer. arXiv preprint arXiv:1904.03711 (2019).

[29] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. Proc. VLDB Endow. 2, 1 (2009), 982–993. https://doi.org/10.14778/1687627.1687738

[30] Parimarjan Negi. 2022. Robust Query Driven Cardinality Estimation under Changing Workloads: Online Appendix. Retrieved 2022 from https://parimarjan.github.io/robust_cardinality_appendix.pdf [Online;].

[31] Parimarjan Negi, Ryan Marcus, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2020. Cost-Guided Cardinality Estimation: Focus Where it Matters. In 36th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2020, Dallas, TX, USA, April 20-24, 2020. IEEE, 154–157. https://doi.org/10.1109/ICDEW49219.2020.00034

[32] Parimarjan Negi, Ryan C. Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. Proc. VLDB Endow. 14, 11 (2021), 2019–2032. https://doi.org/10.14778/3476249.3476259

[33] Andreas Kipf Hongzi Mao Nesime Tatbul Tim Kraska Mohammad Alizadeh Parimarjan Negi, Ryan Marcus. 2021. Cardinality Estimation Benchmark. https://github.com/learnedsystems/ceb [Online;].

[34] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. 2020. QuickSel: Quick Selectivity Learning with Mixture Models. In Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1017–1033. https://doi.org/10.1145/3318464.3389727

[35] Suraj Shetiya, Saravanan Thirumuruganathan, Nick Koudas, and Gautam Das. 2020. Astrid: Accurate Selectivity Estimation for String Predicates using Deep Learning. Proc. VLDB Endow. 14, 4 (2020), 471–484. https://doi.org/10.14778/3436905.3436907

[36] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO - DB2's LEarning Optimizer. In VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy, Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass (Eds.). Morgan Kaufmann, 19–28. http://www.vldb.org/conf/2001/P019.pdf

[37] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In International conference on machine learning. PMLR, 3319–3328.

[38] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. 2011. Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions. Proc. VLDB Endow. 4, 11 (2011), 852–863. http://www.vldb.org/pvldb/vol4/p852-tzoumas.pdf

[39] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? Proc. VLDB Endow. 14, 9 (2021), 1640–1654. https://doi.org/10.14778/3461535.3461552

[40] Lucas Woltmann, Claudio Hartmann, Maik Thiele, Dirk Habich, and Wolfgang Lehner. 2019. Cardinality estimation with local deep learning models. In Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019, Rajesh Bordawekar and Oded Shmueli (Eds.). ACM, 5:1–5:8. https://doi.org/10.1145/3329859.3329875

[41] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation. In SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2009–2022. https://doi.org/10.1145/3448016.3452830

[42] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2022. FactorJoin: A New Cardinality Estimation Framework for Join Queries. arXiv preprint arXiv:2212.05526 (2022).

[43] Ziniu Wu, Amir Shaikhha, Rong Zhu, Kai Zeng, Yuxing Han, and Jingren Zhou. 2020. BayesCard: Revitilizing Bayesian Frameworks for Cardinality Estimation. arXiv preprint arXiv:2012.14743 (2020).

[44] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. Proc. VLDB Endow. 14, 1 (2020), 61–73. https://doi.org/10.14778/3421424.3421432

[45] 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. Proc. VLDB Endow. 13, 3 (2019), 279–292. https://doi.org/10.14778/3368289.3368294

[46] 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. Proc. VLDB Endow. 14, 9 (2021), 1489–1502. https://doi.org/10.14778/3461535.3461539