Robust Query Driven Cardinality Estimation under Changing Workloads

Parimarjan Negi[1], Ziniu Wu[1], Andreas Kipf[1], Nesime Tatbul[12], Ryan Marcus[12], Sam Madden[1], Tim Kraska[1], Mohammad Alizadeh[1]
[1]MIT CSAIL, [2]Intel Labs
{pnegi,ziniuw,kipf,tatbul,rcmarcus,madden,kraska,alizadeh}@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.

1 INTRODUCTION

Cardinality estimators are a critical part of query optimizers [17]. 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 [3, 5, 11, 13, 14, 25, 30, 32–37] use machine learning (ML) models for cardinality estimation — all these methods significantly improve estimation errors, but have different practicality challenges. Moreover, estimation error improvements do not necessarily translate to improved end-to-end query performance [6, 25].

Recent 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 [35, 36], or probabilistic graphical models [5, 11, 30, 34, 37] — 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 larger model sizes and slower inference times. Moreover, these methods also typically do not support all kinds of queries: such as self joins or cyclic joins, or string LIKE filters.

Query driven models [3, 13, 14, 25, 32, 33], on the other hand, do not have these drawbacks. They learn regression models which map queries to their corresponding cardinalities using executed 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 [25, 31], which makes them 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 pure query features with *data features*. For example, past work provides the DBMS estimate of the cardinality as a feature to the neural network [3, 25]. 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 for robustness. As we show, the model may not learn to utilize these features effectively because the query features alone could be enough to perform very well on the training workload.

To avoid this problem, we emulate the characteristics of the workload drift scenarios for which we want robustness *during training*. We can do this by partially masking query information during training (see §4.2). This forces the model to make predictions with missing query information — which emulates the scenario with queries involving new tables or columns. The DBMS estimate would still contain information from the masked part of the query, which the model uses together with the available query information

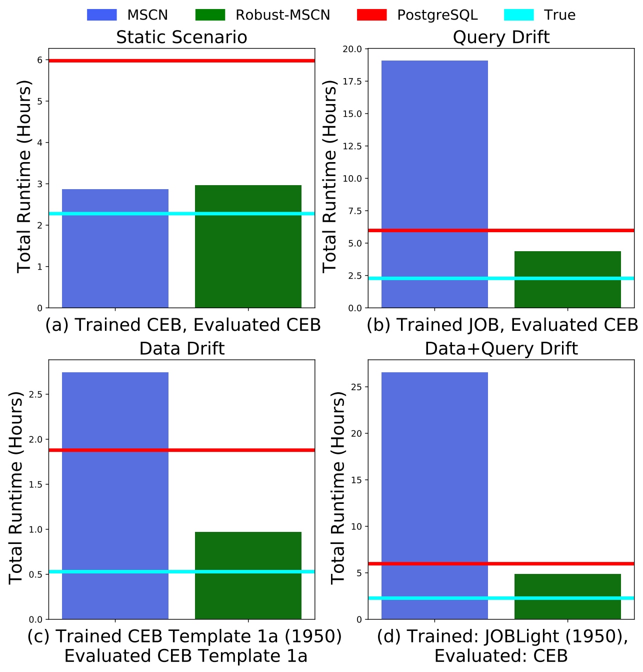


Figure 1: End to end query latencies of the MSCN model vs. our Robust-MSCN model in different workload drift scenarios. Using true cardinalities or PostgreSQL estimates are shown as baselines.

to make predictions. Intuitively, our technique can be viewed as correcting an initial cardinality estimate, provided by the DBMS, using partial information about the query. This anchors the model's predictions to DBMS statistics, which are updated independently of the model, and therefore, meaningful even on queries with new tables or columns. However, the model still uses the query information provided when it helps improve prediction accuracy. For instance, in the training workload, query information can significantly improve performance over using DBMS statistics alone. Our results show that even outside the training workload, the learned model can in some cases use query information to improve accuracy.

Another way to provide the neural network information about the data is through sampling. However, we find that current featurization approaches based on sampling can be particularly misleading in the presence of workload drifts due to the correlations across different table samples induced by joins. We develop a procedure based on sideways information passing [12] to provide sampling features which capture join correlations, and are consistent and useful across workload drift scenarios (see §4.3). Finally, we show how to modify the training procedure, and these techniques, to train a model that adapts to data updates (see §4.4).

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), Cardinality Estimation Benchmark (CEB), and JOBLight-train (see §5.2). 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 includes a subset of our main experiments (§6). In a static scenario (no query/data drift), the state of the art MSCN model [13, 25] 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.

Most 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 §6.4 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 may not require learning complicated joint distributions over all attributes, and effectively utilizing information even from a simple workload can be enough.

To summarize, we experimentally highlight the workload drift challenge across many different scenarios where current query driven models fail. We develop techniques to effectively address these challenges, improving the robustness and practicality of query driven cardinalty estimators. In §2, we give an overview of the ideas developed in other papers that we build upon. §3 provides simple experimental examples that motivate the problem. §4 describes our technical contributions. §5 gives details about the query driven model and the workloads used in the experiments in §6.

2 BACKGROUND

In this section, we will summarize the concepts used in query driven models that we will require when describing our contributions.

2.1 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. [4]. 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*, usually by hashing [3, 25], or *implicitly*, for e.g., by the effect of the filter on a sample.

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. [14]. 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 equivalent SQL query, for instance, filters on different columns may actually have very similar effects. With join bitmaps (§4.3), we will extend this principle to represent similar filters that may be present on different tables.

2.2 Evaluation Functions

Given a query, an optimizer requires cardinality estimates for its subplans, i.e., intermediate joins that it encounters when costing alternative query plans. Using these estimates, the optimizer finds the cheapest query plan. Our goal is for the optimizer to generate good query plans. This leads to a few ways to evaluate how good are the estimates for a query, $\hat{y}$, as compared to the true values, $\boldsymbol{y}$.

a) **Q-Error.** $max(\frac{y}{\hat{y}}, \frac{\hat{y}}{y})$. This is referred to as the Q-Error, or multiplicative error, and is the typical estimation error used in cardinality estimation since Moerkotte et al. [23] showed that it can be used to bound the worst case plan costs.

b) **Query Latency.** To compute this, we inject the estimates, $\hat{y}$, into a DBMS optimizer, get the query plan, and execute it. The query latency numbers should be interpreted in relation to the baselines: the latency for the DBMS plan using true values, $\boldsymbol{y}$, or PostgreSQL estimates.

c) **Relative DBMS Plan Cost.** Defined precisely in [25]. It injects $\hat{y}$ into a DBMS optimizer, get its query plan output, and costs it using true cardinalities, $\boldsymbol{y}$. This is a proxy for latency using cost model units, which can be computed quickly. The cost model, and actual query latencies, don't always align — but it serves as a useful and easy-to-compute signal which we can use to tune our learned models, or analyze their behavior.

The ultimate goal is to optimize for query latencies. Even though Q-Error is a useful measure for query optimization, recent works have shown how the plan cost and query latencies are more correlated [6, 25]. Therefore, we use the DBMS plan cost metric when analyzing the behaviour of learned models, such as learning curves through the training epochs — it is impractical to execute the query plans at each step, and the plan costs are good enough to give us an intuition about how well the model is learning. In the experiments, we divide the DBMS plan cost with the optimal plan cost, i.e., the cost of the DBMS plan for $\boldsymbol{y}$, in order to get a relative metric.

2.3 Loss Functions

The loss function is used to optimize a query driven model, like a neural network, using gradient descent. Therefore, it needs to be differentiable. The query latency, or DBMS plan cost evaluation functions are clearly not differentiable. Therefore, the most popular loss function is Q-Error, used in almost all query driven learned model works [3, 13, 14, 24, 32, 33].

Flow-Loss. An alternative, proposed in [25], is Flow-Loss, which is a differentiable approximation of the DBMS plan cost. This requires it to model the end to end query optimization process using approximations and differentiable primitives — which require several assumptions. Most critically, the simple differentiable cost model used needs to be hand tuned to match the DBMS cost model. If the differentiable cost model is a reasonable approximation to the execution environment, then this loss function leads to more robust models, as argued in [25]. The authors showed that models trained with Flow-Loss generalize better to new templates from the same workload. This is an example of query workload drift, as we discuss in this paper — however, we consider much more extreme drifts as described in §5.2. The approach described in this paper is complementary to Flow-Loss — it can be used to improve Flow-Loss trained models further (see §6.5). 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.

3 MOTIVATION AND INTUITION

In this section, we show a few experiments that convey the key benefits of query driven models, the workload drift challenge, and the intuition behind our training framework for query driven models.

Workloads. To directly compare the query driven and data driven models, we use the STATS-CEB benchmark [6]. Its key benefit is that the queries are supported by all data driven models. However, the workload itself is not very complex: most queries are small (< 4 tables), and PostgreSQL estimates are reasonably good, with the 50th, 90th, and 99th percentile Q-Errors being: 1.5, and 10 and 760. Meanwhile, IMDb workloads like JOB or CEB contain several $10\times$ larger queries (in terms of number of subplans for which estimates are required), and the PostgreSQL Q-Errors have a median of $50 - 100$, with tail Q-Errors in the tens of thousands. The Q-Errors of PostgreSQL estimates are important because these estimates are provided as features to query driven models. Therefore, the IMDb workloads are better for stress testing the query driven models. But,

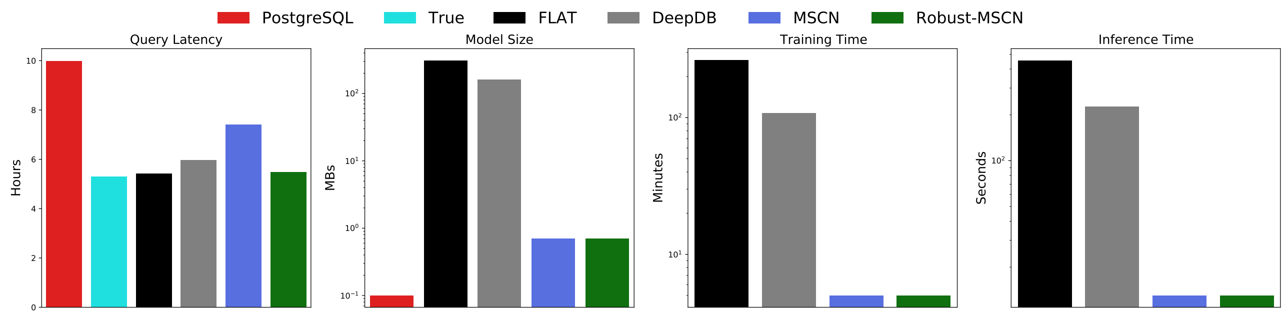


Figure 2: Comparing data-driven models with MSCN and Robust-MSCN across a few modalities relevant to cardinality estimation for query optimization.

the IMDb workloads contain LIKEs, self joins, and cyclic joins — which are not supported by the data driven models at present, therefore, we do not directly compare against them for the experiments on those workloads.

3.1 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. In this section, we compare state-of-the-art query driven and data driven models directly in order to motivate our extensive study on the robustness of query driven models.

Training workload. Since STATS-CEB does not have a training workload, we generate a synthetic query workload with at most four tables per query over STATS-CEB using an automated generator provided in [10]. This is an example of a query workload drift, and is therefore a challenging setting for query driven models.

Figure 2 shows performance of several models on STATS-CEB in terms of end to end query latency, model sizes, and inference time.

Latency performance. The data driven models: FLAT [37] and DeepDB [11] are close to optimal — i.e., the total query latency of plans generated using true cardinalities. The query driven MSCN [13] model still improves over PostgreSQL, and our Robust-MSCN model, which benefits from the techniques discussed in §4, is almost as good as the data driven models despite it being a workload drift scenario.

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 and inference time than the query driven models. This is despite STATS-CEB being a relatively small DB — with total size < 100 MB.

Training time. Similar to model complexity, the training time is much shorter for query driven models as well. But the query driven models require ground truth data for the training workload — which makes the overall training time similar. In practice, however, the ground truth data may be collected from existing logs, thus, the faster training time is another useful property of query driven models.

If we were confident that a query driven model could reliably improve over PostgreSQL with workload drift, as even the MSCN model does on STATS-CEB, then the other benefits: supporting all kinds of queries, low model complexity, faster training and inference times, would make such models very attractive. Next, we will highlight why this isn't always the case with the current techniques of training query driven models.

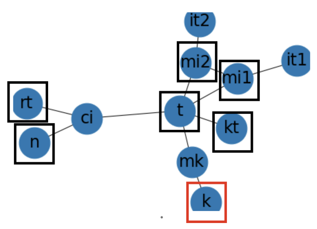
3.2 The Workload Drift Problem

Workload drift is defined precisely in §4.1, and our main experiments in Section 6 explore several differences between the training and evaluation (testing) queries. Here we will use a carefully selected training / evaluation query set where it is possible to understand exactly what is being learned by the query driven model, and develop intuition about our proposed training framework.

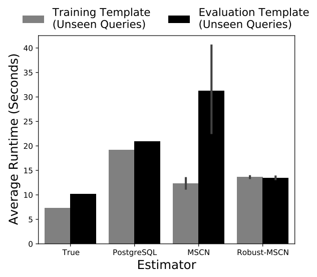
Training / Evaluation workload. We take two templates in CEB that are very similar to each other — one has filters on the column keyword.keyword (template 2b), and one doesn't (template 2a). Each template has a few hundred queries, with filters on several other columns which are generated using the same rules for both the templates. The join graph for queries from this template is shown in Figure 3a. The goal of a cardinality estimation model is to estimate cardinalities of each subplan, i.e. join query involving a subset of the tables in the full query, which are then used by the query optimizer to find the join order to execute the query. There are 290 such subplan estimates in this template.

Effect of the different filter. Intuitively, the training workload and the evaluation workload are very close to each other since almost all the filters are from the same distribution. There is an additional filter on the table 'keyword' in the evaluation queries. This selects for titles with particular keywords, such as 'marvel', or 'superheroes'. Any subplan that involves 'keyword' will naturally get smaller in the evaluation queries. This represents a consistent change from the training workload in all the queries from the new template. Note that the data features — PostgreSQL estimates of the cardinalities — will reflect this change: i.e., the estimate for any subplan with a filter on keyword will have lower cardinality than the estimate for the same subplan without a filter on keyword.

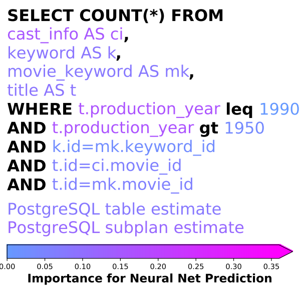
MSCN model is brittle. On new queries from 2b, the training template, MSCN outperforms PostgreSQL. However, on the unseen template, this same MSCN model has a lot of variance across three



(a) Join graph for all queries.



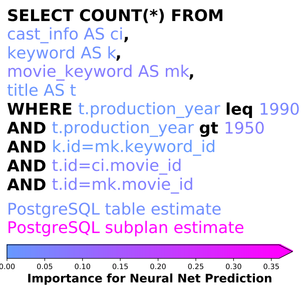
(b) Query performance.



```
SELECT COUNT(*) FROM
cast_info AS ci,
keyword AS k,
movie_keyword AS mk,
title AS t
WHERE t.production_year leq 1990
AND t.production_year gt 1950
AND k.id=mk.keyword_id
AND t.id=ci.movie_id
AND t.id=mk.movie_id
PostgreSQL table estimate
PostgreSQL subplan estimate
```

Importance for Neural Net Prediction

(c) MSCN's importance.



```
SELECT COUNT(*) FROM
cast_info AS ci,
keyword AS k,
movie_keyword AS mk,
title AS t
WHERE t.production_year leq 1990
AND t.production_year gt 1950
AND k.id=mk.keyword_id
AND t.id=ci.movie_id
AND t.id=mk.movie_id
PostgreSQL table estimate
PostgreSQL subplan estimate
```

Importance for Neural Net Prediction

(d) Robust-MSCN's importance.

Figure 3: The MSCN and Robust-MSCN model trained on CEB template 2b, and evaluated on unseen queries from 2b or template 2a, that only differs by one filter on column, keyword, highlighted in red. (b) Shows the runtime performance of baselines, MSCN and Robust-MSCN models. (c-d) Shows the importance of each feature to each model's output when applied on an example subplan.

runs — going from being similar to PostgreSQL to being many times worse than PostgreSQL.

Robust-MSCN model stably improves in both scenarios. Robust-MSCN uses the same training data and features as the MSCN model, but is trained with the query masking technique described in §4.2. Intuitively, it puts more importance on the data features while training. Across all three runs, it improves clearly over PostgreSQL on both the 2b (training template), and 2a (workload drift template).

How different are the MSCN and Robust-MSCN models? We apply interpretablity techniques developed to understand what a deep neural network learns to the MSCN and Robust-MSCN model, and find that they use the input features very differently. We use the integrated gradients algorithm [29] implemented in the Captum library [15]. For a given input, the algorithm considers a neural network's gradient activations for several inputs that interpolate from all zeros to the actual input. Then, it analyzes the gradients to quantify how important each feature was for the model's observed output. We show this analysis for an example subplan of a query from the unseen template; the importance of the input features for the MSCN model is shown in Figure 3c, and the importance of features for the Robust-MSCN model in Figure 3d.

This gives us insight into what the two models learned — and we find that they are actually quite different despite having the same training workload and input features. With the query masking approach, the importance for the PostgreSQL estimate is the highest — i.e., it relies more on that feature. In the standard approach, it relies more on the query features, such as the tables and joins, which is less robust when the evaluation workload changes. Importance attributions over other subplans also show the same pattern. As we discussed above, the effect of the workload drift in this example is coarsely captured by the lower DBMS estimates — and relying more on this feature lets the Robust-MSCN model adapt better to this change. At the same time, the Robust-MSCN model clearly improves over PostgreSQL — thus, it is effectively utilizing the information it learned from the training workload, and not just using PostgreSQL estimates.

4 ROBUST TRAINING FRAMEWORK

4.1 Overview

Workload drift problem statement. 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:

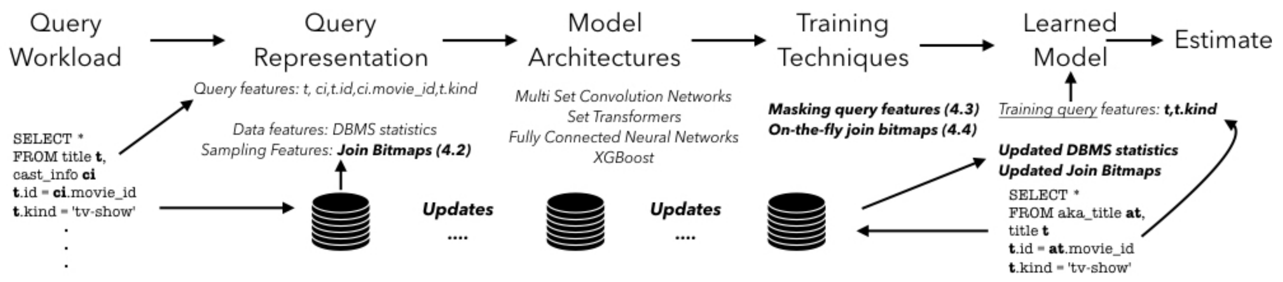
a) New queries have same templates, and only the filter constants change.
b) New queries include filters on new columns, new join graphs (new combination of tables), or entirely new tables.
c) New queries, and their cardinalities, are on updated data.

Several prior works have shown that different query driven models are very effective in the first scenario [3, 14, 25], but these models do not handle either of the second or third scenario well [25, 31]. 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 data features to have reasonable estimates for the rest.

An orthogonal approach to improve a query driven model in the presence of workload shift is to retrain it periodically. However, 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 very simple pairs of training and evaluation workloads on IMDb. We


Query Workload
Query Representation
Model Architectures
Training Techniques
Learned Model
Estimate
Query features: t, ci,t.id,ci.movie_id,t.kind
Data features: DBMS statistics
Sampling Features: Join Bitmaps (4.2)
Multi Set Convolution Networks
Set Transformers
Fully Connected Neural Networks
XGBoost
Masking query features (4.3)
On-the-fly join bitmaps (4.4)
Training query features: t,t.kind
Updated DBMS statistics
Updated Join Bitmaps
Updates
....
Updates
....

```
SELECT *
FROM title t,
cast_info ci
t.id = ci.movie_id
t.kind = 'tv-show'
.
.
.
```

```
SELECT *
FROM aka_title at,
title t
t.id = at.movie_id
t.kind = 'tv-show'
```

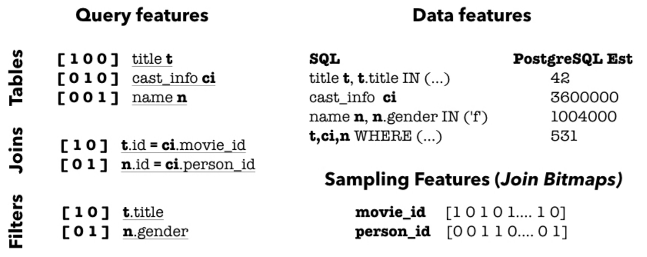
Figure 4: System overview. Our contributions are highlighted, with pointers to their Section numbers.

```
SELECT COUNT(*) FROM
title AS t,
cast_info AS ci,
name AS n
WHERE t.id = ci.movie_id
AND ci.person_id = n.id
AND t.title IN (TITLES)
AND n.gender IN (GENDER)
```

(a) Training template

```
SELECT COUNT(*) FROM
title AS t,
cast_info AS ci,
name AS n
WHERE t.id = ci.movie_id
AND ci.person_id = n.id
AND t.kind = 'movies'
AND n.gender IN ('m', 'f')
```

(b) Example evaluation query



[1 0 0]	title t
[0 1 0]	cast_info ci
[0 0 1]	name n
[1 0]	t.id = ci.movie_id
[0 1]	n.id = ci.person_id
[1 0]	t.title
[0 1]	n.gender

SQL	PostgreSQL Est
title t, t.title IN (...)	42
cast_info ci	3600000
name n, n.gender IN ('f')	1004000
t,ci,n WHERE (...)	531

movie_id	[1 0 1 0 1.... 1 0]
person_id	[0 0 1 1 0.... 0 1]

(c) Query representation for a sample query of the template in (b).

Figure 5: Running example. Queries in the training workload look like (a), and (b) Shows an example of a query with workload drift. (d) Shows the features our scheme uses.

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. For instance, consider Figures 5a and 5b. In reality, the training workload will contain several query patterns, and the differences with the evaluation workload may be more complex — however, such simple cases will allow us to illustrate all our main ideas.

Figure 4 provides an overview of our training framework. Note that the model is fixed after training. We make the model robust to query workload drift by emphasizing the reliable data features during training, and providing more robust sampling features with join bitmaps. We make it more robust to data workload drift by providing it more up to date information in the form of data features and sampling features that are based on the latest state of the database. Below, we will describe each contribution independently.

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. In the following, we first present the drawback of existing approaches and then explain our query masking training scheme.

Query vs. Data Features. When the workload shifts, query features may not be reliable anymore. In particular, there may be queries with a new table, join, column, or constant — such queries may be very close to the training queries, but there would be no meaningful way to encode the unseen query elements. Note, it is possible to have enough spots in a one-hot vector for every table or column in the database, but it will still not be useful for the neural network model, because the model could have learnt to utilize that table/column feature only if it was present in the training workload. In contrast, we always expect to have meaningful data features even in new contexts because data features are computed using reliable and up-to-date DBMS statistics. At the same time, the DBMS makes several simplifying assumptions, and ignores correlations, which can be learned using query features in a workload.

Example. Let us consider a situation that illustrates potential problems with query features when workloads shift. The training workload (Figure 5a) contained filters selecting for specific movies, so it is likely that the size of 'title', and therefore, 'title ⋈ cast_info ⋈ name', was always relatively small in the training workload. Assume there is a new evaluation query with a filter on a different column of title, such as 'title.kind = movies'. The query features do not change much, but it would drastically increase the cardinalities of all sub-plans in the query, since the size of the 'title' table after applying the filter would be much larger. The DBMS estimates for title, and all subplans containing title, such as 'title ⋈ cast_info ⋈ name', would reflect that the new query should be much larger than the ones in the training workload, since they are computed using DBMS statistics and do not depend on the training data.

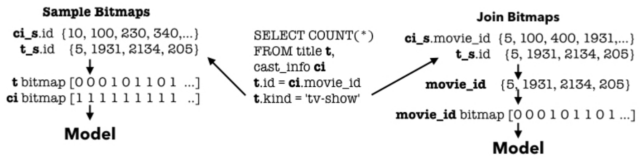
At the same time, the model may have simply learned to map title, or 'title ⋈ cast_info' to relatively small values, and might not utilize the data features effectively. This is because it may have been enough to rely on query features — in this case, the presence of table 'title', which was always small in the training workload — to achieve high accuracy on the training data.

Masking technique. Based on these intuitions, 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 predicting cardinalities with missing information about one or two columns, or tables, which is exactly the scenario we want it to be robust against. The results are not very sensitive to the exact value of p, and we use $p = 0.2$ — so for example, in a five table join, information about one table may be missing. At inference 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.

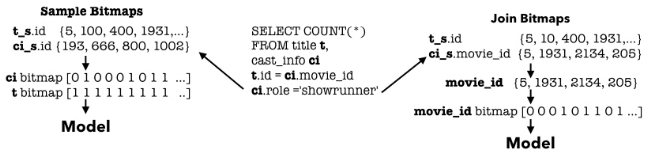
This is similar to a common technique, known as dropout, in ML, although the motivation behind it is different here. Typically, it is used on all input features, or on the hidden layers. It has been shown to be empirically useful for regularization [2]; But in these tasks, such as in vision, often the models serve as feature extractors, and there isn't any inherent structure in the input utilized by them. Nevertheless, the ML dropout use cases show that it is not unreasonable to ask a model to make predictions with some missing information.

As an effect of the dropout technique, the model should rely more on the data features. For instance, in the example above, the model should be able to learn to adjust its estimate upward despite not seeing a filter on title.kind before or seeing any large estimates, just because the DBMS estimates are larger. One might argue that if the model had seen examples of both large and small values for title ⋈ cast_info, then it could have learned to adjust its estimate upward without prioritizing the data features. However, in practice, such effects may appear in complex situations, and it is unreasonable to expect enough coverage of all such situations in the training query workload.

4.3 Join Bitmaps


```
SELECT COUNT(*)
FROM title t,
cast_info ci
t.id = ci.movie_id
t.kind = 'tv-show'
```
Sample Bitmaps
ci_s.id {10, 100, 230, 340,...}
t_s.id {5, 1931, 2134, 205}
t bitmap [0 0 0 1 0 1 1 0 1 ...]
ci bitmap [1 1 1 1 1 1 1 1 1 ..]
Model
Join Bitmaps
ci_s.movie_id {5, 100, 400, 1931,...}
t_s.id {5, 1931, 2134, 205}
movie_id {5, 1931, 2134, 205}
movie_id bitmap [0 0 0 1 0 1 1 0 1 ...]
Model

(a) Example training query.


```
SELECT COUNT(*)
FROM title t,
cast_info ci
t.id = ci.movie_id
ci.role ='showrunner'
```
Sample Bitmaps
t_s.id {5, 100, 400, 1931,...}
ci_s.id {193, 666, 800, 1002}
ci bitmap [0 1 0 0 0 1 0 1 1 ...]
t bitmap [1 1 1 1 1 1 1 1 1 ..]
Model
Join Bitmaps
t_s.id {5, 10, 400, 1931,...}
ci_s.movie_id {5, 1931, 2134, 205}
movie_id {5, 1931, 2134, 205}
movie_id bitmap [0 0 0 1 0 1 1 0 1 ...]
Model

(b) Example evaluation query.

Figure 6: Comparing operations of sample bitmap vs. join bitmap.

Sample bitmaps (see §2) reflect the attribute correlation within each table, but can not capture the correlation across tables induced by a join because it uses independent samples from each table. It does not utilize the fact that the primary key and foreign key refer to the same attribute in different tables.

Join bitmap. Figure 6a 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.

Join bitmap benefits. There are two benefits of this approach. First, the model does not have to learn that the primary key and foreign key columns are equivalent — we can encode this as part of the query representation. Thus, some of the work required for learning is done by utilizing domain knowledge, and not requiring training data.

Second, and more importantly, the sample bitmap approach can be misleading when workload drift occurs. Let's assume all training queries were similar to Figure 6a, with different filters on title. If all the evaluation queries were like that as well, then the sample bitmap approach would work quite well. However, consider Figure 6b, which shows an evaluation query that has a filter on a new table — 'cast_info'. This filter is extremely correlated to the 'title.kind = tv-show' filter from training, since it selects for cast members that are 'showrunners', which occur mostly in tv-shows, thus filtering out a large part of the 'title' table after the join. A sample bitmap approach will create two bitmaps based on the 'title' and 'cast_info' samples. But since filters on cast_info were not seen in the training workload, the model would not have learnt much about the 'cast_info' bitmap. In fact, the sample bitmap approach will actively mislead the model since the title bitmap only has 1s, and in the training data, this would likely correspond to large cardinalities of 'title ⋈ cast_info'. Meanwhile, the consolidated information in the movie_id bitmap presented by the join bitmap approach should capture the induced effect of the filter on 'cast_info' on 'title'. Since the model had seen examples of different 'movie_id' bitmaps in training, this should provide a useful signal for it. This allows us to pass the information about filters across joins — thus, the bitmaps for the primary keys should reflect information that includes the effect induced by filters on tables that have it as a foreign key.

Sampling overhead. This process adds additional overhead at inference time compared to 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 (we use primary key samples of 1000, and foreign key samples of at most $30,000$ — which is less than 1% of the table sizes in IMDb). (2) The number of join columns are more than the number of tables, i.e., more sample tables are needed. For instance, in the workload shown in Figure 5a, the table 'cast_info' will have two sample tables: on 'cast_info.movie_id' and 'cast_info.person_id' (on both the foreign key attributes). But this is a minimal overhead, especially since filters could be applied to the samples in parallel.

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 [1]. 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.

Join bitmap limitations. However, there is no easy way to efficiently capture the correlations across joins involving more than one primary key in a single bitmap. For instance, consider the query

```
SELECT COUNT(*) FROM title as t,  cast_info AS ci,
name AS n WHERE t.id=ci.movie_id AND ci.person_id=n.id
AND n.name ILIKE '%Robert%Downey%'
```

The filter constant on name.name filters out most titles, but this information will not be captured in the join bitmap of title.id. The join bitmap on the primary key 'name.id' should capture it. For this to be a useful signal to the learned model, it requires that the training workload contains coverage of bitmaps on both 'title.id' and 'name.id'. This suggests the requirement for training workloads: we would like to see coverage of bitmaps on all the primary keys involved in the evaluation. Sample bitmaps had provided a unified representation for semantically equivalent SQL queries on a single table. The join bitmaps approach extends this to groups of tables that share semantically equivalent join keys.

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.

Trouble with existing query driven models. In terms of data updates, existing query driven models inevitably recommend collecting new training data (or updating old training queries) by executing the queries, and retraining their models [18–20]. 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 [6, 34].

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 in our framework, we use techniques to make the drop gradual and avoid 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.

Intuitively, this is because we make the model rely on the DBMS estimates more, which are frequently updated along with data updates, and the join bitmap, which can use updated samples, or be sampled on-the-fly. Thus, using the updated data features and the freshly sampled join bitmaps, our framework can produce relatively effective estimates for the updated data without new training queries. For instance, 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'
```

Clearly, the filters are very correlated. A DBMS estimator would consider these filters independent, and under-estimate the cardinality. Given similar training data, a learned model should be able to learn this correlation pattern; however, assume 10× more data has been added to the database. The DBMS estimate based on the updated statistics would still under-estimate the cardinality by ignoring the correlation, but it would be a much larger estimate since the sizes in the base tables would have increased. A learned model may not have needed to use the DBMS estimate to correctly predict cardinalities on the original query / DB pair; however, a model that relies more on the data feature, could have learnt simpler rules that estimate its cardinality w.r.t. the data feature, e.g., learning to adjust the DBMS estimate upwards when such a correlation pattern was present. Such a model would be able to effectively estimate the size despite the 10× increase in data. That is, the model would remain useful if the correlation pattern among columns remains similar. If the correlation pattern itself changes along with data updates — then we would not expect the model to be as useful, although its reliance on the DBMS estimate would ensure that it does not simply output outdated cardinality labels.

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 old 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.

Sacrifice performance for robustness. Admittedly, as compared to using a fixed data sample, shuffling 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-data update scenario, we show results without shuffling bitmaps. Further, in §6.4, we show the performance penalty of shuffling bitmaps in a scenario without data update. In general, since data updates are easily notice-able, it is possible to choose a model based on whether it requires robustness to data updates or not.

We empirically evaluate that the models trained under our framework with updated data features and on-the-fly join bitmaps are more robust against data updates. Moreover, the performance declines predictably as data changes — often, still showing non-trivial gains over PostgreSQL. This is unlike the existing query driven models — which degrade dramatically, having up to 3× slower query execution times than models trained using our framework.

5 MODELS AND WORKLOADS

In this section, we will go over different query driven models, and their trade-offs, and the query workloads we use to evaluate our methods.

5.1 Query driven models

In this section, we will show how the features described before 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. One method flattens all the query information into a 1d vector, and then uses standard learning methods, such as XGboost [3], or fully connected neural networks [4, 32?]. 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) The input sizes can get quite large, especially if we use bitmaps, since each bitmap would have to be assigned to different indices. (2) It can not represent self joins that are not seen in the training workload; similarly, it can not add any information about a new table / or column.

Set featurization. Set based architectures used for cardinality estimation include Multi Set Convolutional Networks (MSCN) [8, 13, 14, 25]. 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. Then, a set of these fixed length vectors is processed using a fully connected network; since the set can have different lengths, the output is averaged together. Finally, the output is processed by fully connected layers. We show that we can also adapt the architecture to use more recent set transformer networks, which does not do an average pooling of each set, instead uses a weighted averaging, where the weights are learnt as part of the model.

The set features are more suited for workload drift, such as self-joins or having new table / joins / columns. A new table / column will not have an appropriate query feature, 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, with MSCN as the representative set network.

Adding data features to set networks. The original MSCN architecture [13] 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

In this section 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 evaluate on other workloads to rigorously evaluate the impact of workload drift.

Join Order Benchmark (JOB). JOB [17] is a set of 113 handwritten queries with up to 16 joins. This was released in 2016, and was not meant to be a ML benchmark — therefore, it does not contain a separate training workload. However, several papers have used it as an evaluation workload for query optimization [9, 21, 22, 25].

Cardinality Estimation Benchmark (CEB). CEB [25] uses handcrafted templates, and synthetic query generation rules to generate several challenging large queries, like JOB, but has several thousand queries. In comparison, JOB contains 5 additional tables, but there is non-trivial overlap in the join graphs and columns used in the two workloads.

JOBLight-train. JOBLight-train [14] is a set of synthetically generated $40K$ queries with 3-table joins. It was used to train a model on an accompanying set of 70 evaluation queries that had up to 6 tables in a join. To the best of our knowledge, this workload has never been used to train a model for evaluation on the more complex JOB workload. Since there are only two joins per query, the workload is fairly trivial as an evaluation workload for query optimization. However, it is a useful training workload since it is very simple, has a lot of training data, and contains several differences with the other workloads.

Evaluating data updates. The full IMDb database we use contains data up to 2013 [25]. 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.

6 EXPERIMENTS

Training / Evaluation workload splits. We train our models on JOBLight-train, JOB, or CEB. We use JOB and CEB as the evaluation workloads as they have complex queries where better cardinality estimates clearly improve query plans. When evaluating on CEB, we use the released test set of 509 representative queries [26]. When training on CEB, we use a training set of 2600 queries [26]. When evaluating on JOB, we use all 113 queries. When training and evaluating both on JOB, we use ten different $50 - 50$ splits of training

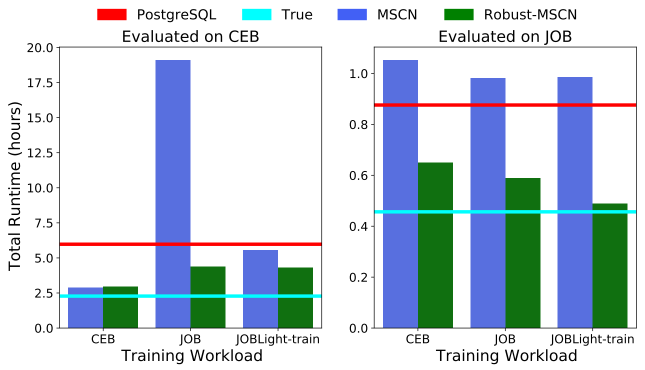


Figure 7: Models trained on CEB, JOB, or JOBLight-train and evaluated with query runtime performance on CEB or JOB.

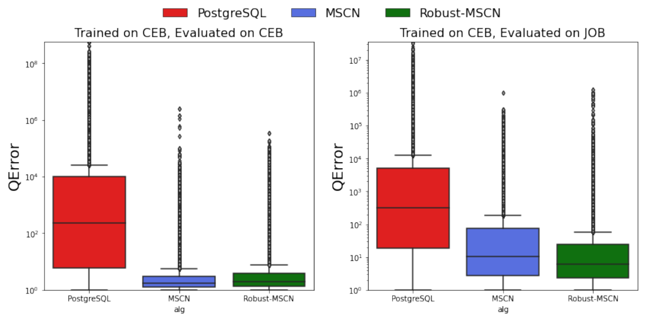


Figure 8: Q-Error of models trained on CEB, and evaluated on new queries from CEB (left) or JOB (right).

and test queries, since the total number of queries are much fewer, so the performance is very sensitive to the differences between the training and evaluation sets. This scenario is an example of a workload drift scenario as well because there are not enough training queries, and each JOB query has a slightly different template — i.e., having filters on different columns etc.

Execution environment. We use the execution environment provided with the Cardinality Estimation Benchmark (CEB) [26]. We use a docker container with PostgreSQL 12, and 1GB RAM. We clear cache before every execution, and use primary and foreign key indexes.

6.1 Cross Workload Generalization

In Figure 7, 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 evaluated on CEB, both the models do clearly better than PostgreSQL, and are very close to the performance of true cardinalities.

MSCN is brittle when evaluated on different workloads. It is expected that workload drift should make a learned model perform 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 a representative subset — models trained on CEB, and evaluated on CEB or JOB — is shown in Figure 8. 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 the MSCN model. However, despite improving Q-Error compared to PostgreSQL, as we saw in Figure 7, the MSCN model has worse runtime. This discrepancy makes sense when you consider that the Q-Errors on JOB are over 113 queries which include $70K$ subplan estimates made by the models — and only improving overall Q-Error estimates is not enough to guarantee better end to end performance in query plans.

6.2 Data Updates

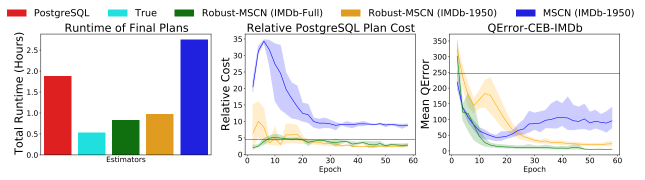


Figure 9: Final latencies, and learning curves for models evaluated on the data drift scenario.

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 9 shows the results of models trained on CEB template 1*a* with ground truth data from IMDb-1950, and evaluated 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 9 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 10 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

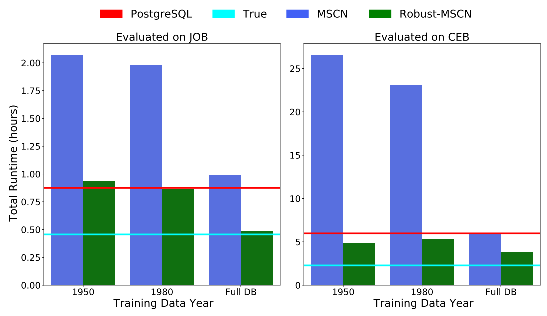


Figure 10: Models trained on JOBLight-train workload with ground truth cardinalities from IMDb-1950, IMDb-1980, or the full IMDb version. The performance of all models are evaluated on the full IMDb version.

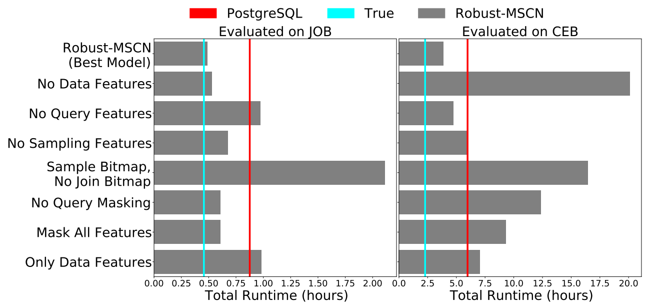


Figure 11: Ablation study. Each label (y-axis) is a difference from the Robust-MSCN model trained on JOBLight-train, evaluated on JOB (left) or CEB (right).

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.

6.3 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 11 shows the results of training a model on JOBLight-train after some key modifications compared to the Robust-MSCN model.

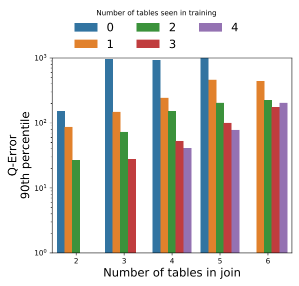
No query features. Throughout our scheme, we have focused a lot on effectively utilizing the data and sampling features. It is natural to ask if the query features are even required? As we see, there is a noticeable degradation in performance without the query features — especially on Q-Errors. This 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' essentially performs the same as 'No Query Masking', and in particular, both these get worse than PostgreSQL when evaluated on CEB.

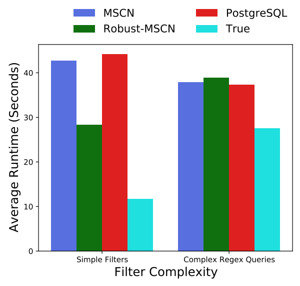
Overlap between sampling and data features. Both the sampling and data features capture information about the underlying data — interestingly, the model still does quite well on JOB if we just remove one of them. However, the performance degrades drastically when both are removed. On CEB, the performance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

6.4 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.



(a) Robust-MSCN Q-Errors.



(b) Simple vs. complex filters.

Figure 12: Exploring where Robust-MSCN trained on JOBLight-train improves.

Analyzing JOBLight-train Q-Errors. Even though we see consistent improvements over PostgreSQL in Q-Errors, typically there is still a long tail of Q-Errors. This is expected: there are new tables, and new join graphs in these workloads that were not in JOBLight-train; Figure 12a shows that the Q-Errors are much worse if no table in the subplan query is present in JOBLight-train; But, when one or more tables from the training workload are present, the Q-Errors improve. This suggests that the model is learning to incorporate information from the simple JOBLight-train workload in a sensible way over more complex queries. This is particularly important in JOB queries because several JOB queries have very extreme filters, for instance: '17f.sql' has 'n.name LIKE "%B%"', or '6e.sql' has 'n.name LIKE "%Downey%Robert%"'. JOBLight-train contains thousands of queries with filters on cast_info.person_id, thus, it could learn to correct the DBMS estimate upward for filters that select a large amount of table, or correct the estimate downward for filters that select for just one movie or actor.

What doesn't improve using just JOBLight-train queries? CEB has many more automatically generated queries than JOB, and samples filters based on their correlations across tables; therefore, it has fewer such extreme filters, which explains why we see much less improvement on it when we train on the simple JOBLight-train. Figure 12b shows that we see no improvement over a subset of the more complex regex templates on CEB, and a slight improvement

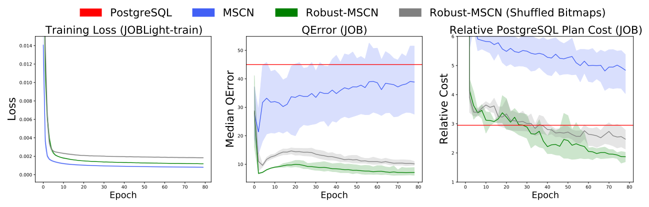


Figure 13: Learning curves for models trained on JOBLight-train. Showing training loss (on JOBLight-train), Q-Error on JOB, and relative plan costs on JOB.

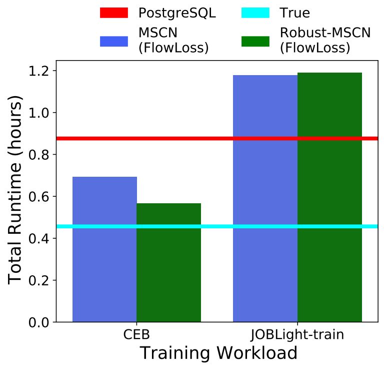


Figure 14: Total latency on JOB for models trained with FlowLoss instead of Q-Error.

over simpler templates that involve categorical filters over attributes such as 'genre', 'language' etc.

Learning curves. Let us consider the learning curves of the MSCN and Robust-MSCN model trained on JOBLight-train (Figure 13). In the first couple of epochs, both the models go from essentially random estimates, to reasonable estimates that improve estimation 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 queries. The Robust-MSCN model, on the other hand, continues to improve Q-Error on JOB throughout training (unlike the MSCN model), therefore, it shows that it is learning generalizable and useful features rather than overfitting to the exact cardinality distributions in the training workload. 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 is able to provide consistent estimates, and avoid noisy and random 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 13, we show the performance penalty we would have if we used the shuffled bitmap approach when there were no data updates.

6.5 Comparison to Flow-Loss

In §2, we discussed the trade-offs between Q-Error or Flow-Loss as the loss function. Figure 14 shows the runtime performance on JOB from training MSCN or Robust-MSCN on CEB using Flow-Loss.

Robust-MSCN is complementary to Flow-Loss. The standard MSCN model also improves over PostgreSQL when trained on CEB; recall from Figure 7 that MSCN with Q-Error had actually got worse. Thus, Flow-Loss also improves robustness to workload drift when the cost model is well tuned. However, using the Robust-MSCN model with Flow-Loss improves the performance further — showing that these are complementary techniques.

Flow-Loss does not work well without complex query workloads. Flow-Loss optimizes for an approximate plan cost; JOBLight-train contains very simple queries, and thus, there are not many query plans. So a Flow-Loss trained model may not be able to learn much there — and its performance would be unpredictable.

7 RELATED WORK

Many early work using ML in DBMS was for improving cardinality estimation [5, 16, 28], and several classical approaches have been proposed for it, but it has remained a challenging and open problem. Recent ML work 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 [35], sum product networks [11, 37], or probabilistic graphical models [30], or other methods [7, 11, 27, 33, 36].

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 [13], lightweight models [3, 4], and several other related variants [3, 8, 9, 14, 25, 32, 33].

Retraining models. Typically, query driven approaches recommend retraining models in case of workload drift. Warper [18] was a recent system built around this idea: it automatically detects workload drift, generates similar new queries, collects ground truth data, and retrains the model. This approach is complementary to the techniques described in our work — we will require retraining models eventually as well, but we want to not require it after every slight change in the workload.

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 [18]. 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.

REFERENCES

[1] 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

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

[3] 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

[4] 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

[5] 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

[6] 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

[7] 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

[8] 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

[9] 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

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

[11] 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

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

[13] 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.

[14] 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

[15] 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).

[16] 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

[17] 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

[18] 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.

[19] 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).

[20] 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.

[21] 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.

[22] 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).

[23] 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

[24] 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

[25] 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

[26] Andreas Kipf Hongzi Mao Nesime Tatbul Tim Kraska Mohammad Alizadeh Parimarjan Negi, Ryan Marcus. 2021. Cardinality Estimation Benchmark. [Online;].

[27] 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

[28] 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

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

[30] 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

[31] 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

[32] 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

[33] 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

[34] 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).

[35] 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

[36] 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

[37] 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