Are Learned DBMS Components Robust to Workload Drift?: [Experiments & Analysis]

ZIZHONG MENG, Nanyang Technological University, Singapore
GAO CONG, Nanyang Technological University, Singapore
SIQIANG LUO, Nanyang Technological University, Singapore

Workload-driven learned database components leverage historical query workloads to build predictive models. These models typically take queries or query plans as input and generate predictions or decisions to optimize query performance. However, the effectiveness of these models heavily depends on the quality and distribution of the training query workload (training workload). While prior studies focus on scenarios where the test query workload (test workload) distribution matches that of the training workload, real-world workloads often change over time, leading to workload distribution drift. In this paper, we explore the impact of workload drift on workload-driven models across different database components. We generate a variety of workload drift scenarios based on IMDb and STATS datasets, and analyze how different levels and types of drift affect estimation performance of each component and overall query execution performance.[1] Our findings highlight the limitations of current workload-driven models under workload drift and provide insights into designing more robust and adaptive solutions.

CCS Concepts: • **Information systems** → **Query optimization**; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: Workload-driven database systems, Workload drift problem

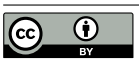
ACM Reference Format:
Zizhong Meng, Gao Cong, and Siqiang Luo. 2026. Are Learned DBMS Components Robust to Workload Drift?: [Experiments & Analysis] . *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 137 (June 2026), 24 pages. https://doi.org/10.1145/3802014

1 Introduction

Workload-driven learned models in database systems replace traditional rule-based components with data-driven models that learn directly from query workloads. Instead of relying on manually designed cost formulas or independence assumptions, these models capture complex correlations between tables and columns observed in historical query executions. For example, learned cardinality estimation models [15, 22, 33] predict query result sizes from SQL query structures, while cost estimation models [28, 42, 53] take query plans as input to predict execution latency or resource usage. These models are typically trained and evaluated under experimental settings with comprehensive and balanced training workloads, where no workload drift occurs. As a result, they generalize well to the prediction workloads within the same distribution. However, this assumption rarely holds in real-world deployments, where the workload distribution often drifts due to changes

[1]The data, source code and experimental results are available on https://github.com/mzz235711/Workload-Drift-Benchmark

Authors' Contact Information: Zizhong Meng, Nanyang Technological University, Singapore, zizhong001@e.ntu.edu.sg; Gao Cong, Nanyang Technological University, Singapore, gaocong@ntu.edu.sg; Siqiang Luo, Nanyang Technological University, Singapore, siqiang.luo@ntu.edu.sg.



ACM 2836-6573/2026/6-ART137
https://doi.org/10.1145/3802014

```
SELECT * FROM title as t,cast_info as ci,movie_info as mi, movie_info_idx as mi_idx
    WHERE t.id=mi.movie_id AND t.id=mi_idx.movie_id AND t.id=ci.movie_id AND
    t.production_year>#1 AND t.production_year<#2 AND ci.role_id>#3 AND ci.role_id<#4
    AND mi.info_type_id>#5 AND mi.info_type_id<#6;
```

Query id	Placeholder values	Cardinality	Execution time
q_1	$\{\#i\}_{i=1}^{6} = \{1898, 1902, 1, 3, 12, 38\}$	213	1s
q_2	$\{\#i\}_{i=1}^{6} = \{1969, 1997, 8, 10, 92, 100\}$	224052	6s
q_3	$\{\#i\}_{i=1}^{6} = \{1960, 2005, 6, 10, 69, 77\}$	224713	16s

Fig. 1. Example query and corresponding placeholder assignments

in query structures. As a result, the performance of learned models can significantly degrade when facing out-of-distribution (OOD) workloads.

Although prior studies [12, 13, 54] have evaluated learned database models under in-distribution (ID) settings using benchmarks such as JOB [18], TPC-H, DSB [6], and STATS-CEB [12], these benchmarks focus on static workloads where training and testing queries share the same distribution. ShiftHandler [47] and Warper [20] reveal that workload drift can reduce model performance, but they only simulate drift by changing query templates, without systematically varying key factors such as join structures, filter predicates, or value distributions. Moreover, they do not quantify how different drift types or magnitudes affect model robustness. Consequently, existing benchmarks cannot comprehensively evaluate the resilience of learned DBMS components under realistic workload changes, highlighting the need for a benchmark that explicitly models and measures workload drift.

In this paper, we propose a new evaluation framework to systematically study the impact of workload drift on the performance of workload-driven learned components. Our goal is to build a comprehensive benchmark that facilitates the community to evaluate model performance under controlled distributional drifts. Specifically, we aim to address the following three research questions:

RQ1. How can we generate different types of workload drift for benchmarking? Existing studies often rely on query template distributions to detect and simulate workload drift [20, 47]. A query template refers to a parameterized SQL query in which predicate values are masked. For example, a query template for IMDb dataset might take the form in Figure 1, where the placeholders $\#i$ can be replaced with concrete values to generate specific query instances. These approaches typically assume that queries derived from the same template exhibit similar structural features and produce similar database component outputs, where the outputs may correspond to cardinalities, costs, or other task-specific metrics depending on the database task. However, this assumption has several limitations. (1) The same template does not imply the same DBMS component output. Queries generated from the same template can exhibit significant difference in their outputs. For example, the cardinalities of q_1 and q_2 in Figure 1 are very different. Conversely, queries generated from different templates can produce similar outputs. (2) DBMS component outputs can be inconsistent across components because outputs are often component-specific. For example, the execution times of q_2 and q_3 in Figure 1 are very different, while their cardinalities are nearly the same. This inconsistency challenges the use of templates as a universal proxy for query behavior. (3) The template-level similarity is neglected. Existing methods typically ignore the degree of similarity between different templates. Compared to queries derived from similar

templates, those with structurally dissimilar templates are more likely to cause larger workload drift.

Due to these limitations, using query template distributions as a proxy for workload drift is not sufficient. In this paper, we design a workload drift generation technique that modifies training and testing workloads along specific dimensions. We identify four types of workload drift:

- **Output Drift.** The DBMS component outputs (e.g., cardinality or cost) in two workloads have different distributions, while the distribution of query structures (i.e., the composition of joins and filter predicates that define query templates) remains the same.
- **Join Drift.** Join predicates of queries in the two workloads have different distributions, while the filter predicates are uniformly sampled.
- **Column Drift.** Join predicates of queries in the two workloads are randomly sampled, while filter columns are sampled from different distributions.
- **Value Drift.** Queries have the same joins and filter columns, while filter values are sampled from different distributions.

For prior work on evaluating workload drift, Warper [20] considers only Value Drift, while ShiftHandler [47] generates drift solely by altering query templates, without decomposing it into Join Drift and Column Drift. In this work, we design two benchmark workloads, JOB-light-drift based on IMDb [18] and STATS-drift based on STATS [12], to simulate realistic workload drift scenarios. Each benchmark incorporates four drift types with varying degrees of complexity, enabling systematic and comprehensive evaluation of learned DBMS components under distributional drifts.

RQ2. How does workload drift affect the estimation performance of workload-driven models? Several proposals [20, 47] have evaluated the performance of learned database components under workload drift and demonstrated that these components suffer from workload drift. However, they do not discuss how different types and complexity of workload drifts affect the performance of database components. To evaluate the impact of workload drift on learned database components, we conduct systematic experiments on representative models using the two proposed benchmark workloads. For each component, we vary both the type and degree of workload drift to identify which types of drift have the most significant impact on component performance. This analysis reveals which components are most sensitive to specific drift types and how drift intensity correlates with estimation accuracy degradation.

RQ3. If workload drift degrades estimation performance, does this translate into worse query execution performance? A decline in model estimation accuracy does not necessarily lead to an increase in query execution time. In some cases, suboptimal estimates may still result in near-optimal query plans. Therefore, to better understand the practical impact of workload drift, we conduct an end-to-end evaluation across different database components and drift types. Specifically, we evaluate how different types and degrees of workload drift affect query execution time by undermining the estimation accuracy of learned models. This allows us to quantify which model performance degradation actually impacts query execution time.

RQ4. Insights for future work: which workload drift to focus on? Building on our evaluation results, we identify scenarios where addressing workload drift leads to significant performance improvements. This analysis (1) informs practical strategies by highlighting which types of workload drift future research should prioritize when designing robust learned database components, and (2) indicates the workload drift scenarios under which learned database components should be retrained.

In summary, our work makes the following contributions:

- We propose four workload drift types: Output Drift, Join Drift, Column Drift, and Value Drift. Based on these drift types, we design two benchmark workloads: JOB-light-drift and

STATS-DRIFT, to systematically study workload drift in learned database systems. To the best of our knowledge, this is the first work to introduce benchmark workloads specifically targeting the workload drift problem.
- We conduct comprehensive experiments on a range of learned database components, including cardinality estimation, cost estimation, and query optimization, to investigate how different types of workload drift affect model performance. Furthermore, we evaluate whether performance degradation caused by workload drift leads to increased query execution time, providing an end-to-end assessment of drift impact.
- By quantifying the effects of workload drift on both estimation performance and query execution efficiency, we provide insights and practical guidelines to design more robust learned database components.

2 Workload Drift in Learned Database Systems

This section first introduces concepts of workload drift. We then discuss different types of workload drift and how to quantify the degree of drift complexity. Furthermore, we present JOB-LIGHT-DRIFT and STATS-DRIFT benchmark workloads.

2.1 Problem Definition

Many workload-driven learned database components assume that prediction workload shares the same query patterns with the training workload. However, the distribution of predication queries usually changes over time in real scenarios, which might result in inaccurate estimation. This is known as workload drift problem (Definition 1).

Definition 1 (Workload Drift in Learned Database Components). *A workload-driven model M is trained for the database component C. Workload drift in learned database systems refers to the phenomenon where the statistical characteristics or distribution of the incoming workload W changes over time from the training data distribution on which the workload-driven models were originally trained.*

2.2 Quantifying the Degree of Workload Drift

Some recent studies have tried to address the problem of workload drift in learned database systems [20, 33, 47]. However, these works do not fully explore how different types of workload drift affect the performance of specific workload-driven database components. They typically assume that workload drift occurs when the distribution of query templates in the prediction workload differs from that in the training workload [20, 47]. Under this view, drift is often detected by comparing the frequency or presence of distinct query templates across workloads.

While this template-based approach can be reasonably applied in some cases, it has several fundamental limitations:

Ignoring Predicate Value Drift. These methods overlook the predicate value distributions within query templates. Two workloads generated from the same query template can still exhibit drastically different behaviors if the predicate values vary significantly. As illustrated in Figure 1, although q_1 and q_2 share the same query template, their cardinalities differ substantially since the predicate values are drawn from distinct regions of the underlying data domains. This demonstrates that predicate value drift is a critical dimension of workload drift that template-based methods ignore.

Output variation of queries from the same template. The similarity between query outputs is often task-dependent, as two queries may yield comparable results for a particular database component (e.g., execution cost) but significantly different outputs for another (e.g., cardinality). For example, queries generated from the same template might yield similar costs, yet their cardinalities

can differ widely depending on the data distribution. This indicates that evaluating drift solely based on outputs from one component fails to capture its broader impact across the system.

Lack of modeling template similarity. Prior work generally treats query templates as discrete and unrelated categories, without considering semantic similarity between them. However, the structural differences between templates can greatly influence model behavior and output distributions. For example, queries derived from templates with identical join structures but differing projection or filter columns tend to share similar execution characteristics. In contrast, templates with significantly different join structures tend to yield dissimilar costs and plans. Therefore, failing to measure template-level similarity can lead to coarse and misleading notions of drift.

In this work, we consider four types of workload drift from both the query structure and output perspectives. Table 1 illustrates the examples of different workload drift types.

Output Drift. In this setting, queries in both workloads are drawn from the same query distribution (i.e., same templates and structures), but their outputs (e.g., cardinality or cost) follow different distributions. To construct such drift, we first generate a large batch of queries using a fixed set of join graphs, and then sample two subsets whose output distributions differ. The degree of output drift is quantified using statistical distance metrics such as KL-divergence between the output distributions of the two workloads. For cardinality estimation, the Output Drift between q_1 and q_2 in Table 1 is greater than that between q_2 and q_3 because the difference in output is larger.

Join Drift. Here, the queries in each workload differ in their join structures, while their filter predicates are randomly generated. To control and measure the degree of drift, queries from one workload are randomly generated from a predefined set of possible join graphs, where all join graphs have same number join predicates. The degree of drift can be approximated by differences over number of joins in the two workloads. The Join Drift between q_1 and q_5 in Table 1 is larger than that between q_1 and q_2 because their join counts differ more substantially.

Column Drift. In this type, the join structures are all randomized across workloads, but the difference lies in the filter columns, i.e., the attributes used in selection predicates. Each workload is generated with a different number of predicates. The degree of drift is estimated based on the number of predicates. The Column Drift between q_2 and q_4 in Table 1 exceeds that between q_2 and q_3 due to a larger difference in the number of predicate columns.

Value Drift. In this setting, both workloads share the same join graphs and predicate columns, but differ in the value distributions used in the predicates. For example, one workload may sample predicate values from the low end of a column's domain, while the other samples from the high end. This creates differences in query selectivity without altering the query structure. The drift degree is quantified by statistical distance between predicate value distributions (e.g., KL-divergence).

Table 1. Examples of workload drift

Query	SQL	Cardinality
q_1	`SELECT * FROM A WHERE A.x>10;`	50
q_2	`SELECT * FROM A,B WHERE A.id=B.id AND A.x>10;`	500
q_3	`SELECT * FROM A,B WHERE A.id=B.id A.x>20 AND B.y>40;`	400
q_4	`SELECT * FROM A,B WHERE A.id=B.id AND A.x>20 AND B.y>40 AND B.z>50;`	300
q_5	`SELECT * FROM A,B,C WHERE A.id=B.id AND B.id=C.id AND A.x>20 AND B.y>40;`	5000

Table 2 summarizes the four types of workload drift, each of which is introduced at a different level of query granularity. For each drift type, the degree of drift can be systematically controlled.

Table 2. Comparison of the four workload drift types

Drift Type	Join Graph	Pred. Cols.	Pred. Vals.	Outputs
Output Drift	Random	Random	Random	Diff. distribution
Join Drift	Diff. #joins	Random	Random	Induced by queries
Column Drift	Same	Diff. #pred. cols	Random	Induced by queries
Value Drift	Same	Same	Diff. dist.	Induced by queries

Comparison with existing workload drift simulation techniques. Prior techniques simulate only a subset of the drift types considered in this work, and often treat structural drift as a single coarse category. Robust-MSCN [33] covers (1) changing filter constants within the same templates (our Value Drift) and (2) introducing predicates on unseen columns or unseen join graphs, but it does not distinguish the latter into Column Drift and Join Drift. Similarly, ShiftHandler [47] modifies query templates to induce drift (aligned with Column Drift and Join Drift), yet it also does not distinguish these two structural changes. In contrast, Warper [20] focuses solely on Value Drift by sampling filter constants from different distributions, without modeling structural changes in join graphs or predicate sets. Overall, our framework subsumes the drift types in prior work and provides a more comprehensive, fine-grained decomposition, enabling systematic analysis of how different drift forms affect database components. Our results in Sections 4–6 further show that this finer modeling exposes distinct behavioral changes of learned and traditional components that are obscured by coarser drift simulations.

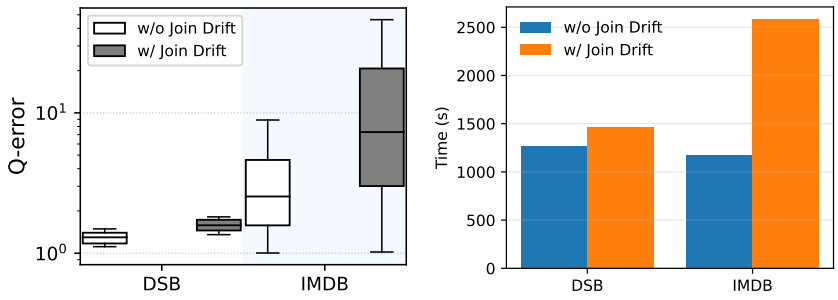
2.3 JOB-light-drift and STATS-drift

We generate two workload drift benchmarks based on JOB-light [15, 18] and STATS-CEB [12] respectively for the following reasons.

To the best of our knowledge, no public dataset provides both real data and real query logs. This fundamental limitation prevents a fully realistic yet reproducible study of real workload drift. Our survey shows that industrial traces (e.g., Microsoft [7] and Google [40]) are not publicly available; Snowset [44] releases query logs without the underlying data; and widely used public datasets (e.g., US Census [43], Flight Delay [35], and NYC Taxi [34]) provide real data but lack real query workloads. In contrast, TPC-style benchmarks (TPC-H/TPC-DS/DSB) [6] rely on synthetic data and synthetic workloads, and thus cannot reflect real workload evolution.

Among publicly available benchmarks, JOB (including JOB-light) and STATS-CEB are the closest approximations to real workloads. They pair real-world datasets with real-workload query templates that capture the semantics of production analytics [12, 18]. Their queries are instantiated from manually designed templates, rather than generated arbitrarily, thereby preserving realistic join structures, predicate patterns, and analytical intents. This template-based paradigm is widely regarded as the best available proxy in the absence of public production traces. Accordingly, we build our workload-drift benchmarks on JOB-light and STATS-CEB to retain semantics while ensuring reproducibility.

IMDb and STATS exhibit data characteristics representative of production systems and are inherently sensitive to workload drift. Compared with common benchmarks such as TPC-H, TPC-DS and DSB, IMDb and STATS are more drift-sensitive due to pronounced skewness and strong column correlations. Figure 2 illustrates cost estimation and query execution performance under



(a) Cost estimation (b) Query execution time

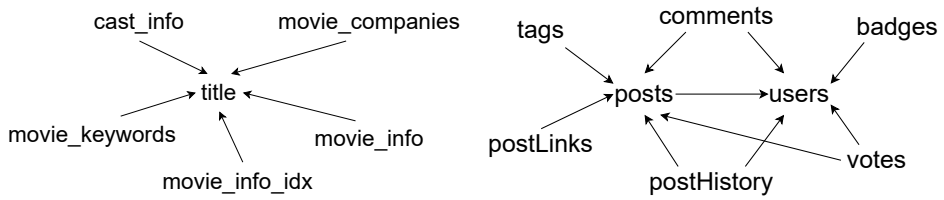
Fig. 2. Performance of DBS and IMDb under join drift

Join Drift on DSB and IMDb: performance on DSB remains almost unchanged as drift increases, whereas IMDb degrades substantially. This suggests that DSB lacks the skewness/correlation patterns that typically make real-world workloads challenging and susceptible to drift. In contrast, IMDb exhibits moderate skewness with meaningful correlations, while STATS shows higher skewness and stronger correlations. We quantify correlations using Nonlinear Correlation Information Entropy (NCIE) [46] and distributional skewness using Mardia's Multivariate Entropy (Mardia) [56] (Table 3); NCIE ranges in $[0, 1]$ (smaller implies weaker correlation), and Mardia values closer to 0 indicate distributions closer to multivariate Gaussian. Overall, STATS exhibits stronger correlations and higher skewness than IMDb, reflecting properties widely observed in production databases where workload drift most strongly affects learned components. Together, IMDb and STATS cover a broad spectrum of realistic distributions and correlation structures, enabling systematic workload drift evaluation.

Table 3. Correlation and skewness of IMDb and STATS

	IMDb						STATS							
	ci	mc	mi	mi_idx	mk	t	b	c	ph	pl	p	t	u	v
NCIE	0.16	0.22	0.18	0.45	0.39	0.21	0.46	0.33	0.31	0.34	0.16	0.04	0.26	0.35
Mardia	11.7	7.2	9.9	2.7	12	276	8.6	120	74.4	34.9	389	714	293	8434

Hence, we design two benchmarks: JOB-LIGHT-DRIFT derived from JOB-light workload and IMDb dataset [15, 18], and STATS-DRIFT derived from STATS-CEB [12] workload and STATS dataset. We do not design workload drift based on the JOB workload, which involves more tables than JOB-light, because our objective is to vary the complexity of workload drift rather than the complexity of individual queries. To control the degree of join drift, we gradually increase the number of joins in the training workloads. Both benchmarks are generated using the same methodology, and each includes four types of workload drift: Output Drift, Join Drift, Column Drift, and Value Drift. Table 4 summarizes workload drift construction on JOB-LIGHT-DRIFT and STATS-DRIFT. We follow the same query-generation procedure as JOB-light for JOB-LIGHT-DRIFT and STATS-CEB for STATS-DRIFT. Specifically, JOB-LIGHT-DRIFT uses the same query templates as JOB-light, and STATS-DRIFT uses the same templates as STATS-CEB. These templates are manually selected, semantically meaningful, and designed to reflect realistic analytical workloads. Figure 3 demonstrates join graphs of JOB-LIGHT-DRIFT and STATS-DRIFT. We follow the same instantiation procedure as JOB-light and STATS-CEB so that all queries are generated from the same template candidate set with JOB-light



(a) JOB-light-drift (b) STATS-drift

Fig. 3. Join graphs of benchmark workloads

and STATS-CEB, and inherit the same semantics. The only modifications we introduce in generation are controlled changes on top of the standard template-instantiation procedure:

For Output Drift, we generate 4,000 queries, where all queries are uniformly sampled from the full candidate template set, and their predicate values are sampled uniformly from the valid value domains. All queries are partitioned evenly into two workloads based on their output values under a given task. For instance, in cardinality estimation, the output corresponds to query cardinalities, while in query optimization, the output corresponds to execution time. Since output definitions differ across tasks, the resulting partitions are task-specific. For evaluation, we adopt a bidirectional setting: the workload with smaller outputs is treated as the training workload and the one with larger outputs as the testing workload, and vice versa.

For Join Drift, we construct multiple workloads, each containing 2,000 queries with a fixed number of joins. To generate a query with i joins, we uniformly sample from templates that contain exactly i joins, then uniformly sample meaningful predicate values to instantiate the template. In JOB-light-drift, queries with 2 joins are the test workload and queries with 0–4 joins are individually as training workloads . In STATS-drift benchmark, queries with 3 joins are the test workload, and queries with 1—5 joins are individually as training workloads.

For Column Drift, we similarly generate multiple workloads, each containing 2,000 queries with a fixed number of filter columns. To generate a query with i predicate columns, we uniformly sample from templates that include exactly i predicates and then uniformly sample meaningful predicate values. In JOB-light-drift, workloads with 1—7 columns are used individually as training workloads, and the workload with 4 columns is the test workload. In STATS-drift, workloads with 1—9 columns are used individually as training workloads, and the workload with 5 columns is fixed as the test workload.

For Value Drift, we generate queries using fixed query templates to eliminate structural variation, where join structures and filter columns are held constant across workloads, while predicate values are sampled from different Beta distributions. This creates workloads that differ only in value distributions and thus in selectivity. The query template for JOB-light-drift is:

```
SELECT * FROM title as t, cast_info as ci, movie_info as mi, movie_info_idx as mi_idx
    WHERE t.id=mi.movie_id AND t.id=mi_idx.movie_id AND t.id=ci.movie_id AND
    t.production_year># AND t.production_year<# AND ci.role_id># AND ci.role_id<# AND
    mi.info_type_id># AND mi.info_type_id<#;
```

The query template for STATS-drift is:

```
SELECT * FROM votes as v, posts as p, badges as b, users as u WHERE p.Id = v.PostId
    AND u.Id = p.OwnerUserId AND u.Id = b.UserId AND p.Score># AND p.Score<# AND
    p.CommentCount># AND p.CommentCount<# AND u.DownVotes># AND u.DownVotes<# AND
    v.CreationDate># AND v.CreationDate<# AND v.VoteTypeId># AND v.VoteTypeId<# AND
    b.Date># AND b.Date<#
```

We generate 8 workloads by sampling predicate values from Beta(7, 2), Beta(5, 2), Beta(3, 2), Beta(2, 2), Beta(1, 1), Beta(2, 3), Beta(2, 5) and Beta(2, 7) for both benchmarks. The workload sampled from Beta(7, 2) is designated as the test workload, while all 8 workloads are used as training workloads in turn to evaluate different levels of drift.

Table 4. Summary of workload drift construction in JOB-LIGHT-DRIFT and STATS-DRIFT

Drift Type	Benchmark	Generation Rule	Test Workload	Training Workloads
Output Drift	Both	4k random generated queries	Small/Large-output queries	Large/Small-output queries
Join Drift	JOB-LIGHT-DRIFT STATS-DRIFT	2k queries per join count	Queries with 2 joins Queries with 3 joins	Queries with 0—4 joins Queries with 1–5 joins
Column Drift	JOB-LIGHT-DRIFT STATS-DRIFT	2k queries per column count	Predicates on 4 columns Predicates on 5 columns	Predicates on 1—7 columns Predicates on 1—9 columns
Value Drift	Both	2k queries per Beta(α, β) distribution	Queries from Beta(7, 2)	Queries from all 8 Beta distributions

3 Evaluation Methodology

In this section, we first discuss the selection of database components for evaluation. Afterwards, we explain the evaluation setup.

3.1 Selected Database Components

Many database components are influenced by workload drift. In this work, we focus on three representative tasks: **cardinality estimation (CE)**, **cost estimation (CostE)**, and **query optimization (QO)**. These tasks cover both query-level representations (CE) and query plan representations (CostE and QO), and they are commonly studied in prior research.

CE aims to predict the result size of a query. Workload-driven cardinality estimators typically encode SQL queries as input features and predict cardinalities. Numerous other database tasks also rely on query representations, including workload generation [48] and approximate query processing [25]. We select three representative methods for evaluation: one early classic approach MSCN [15], and two more recent models, ROBUST-MSCN [33] and ALECE [22].

MSCN [15] builds a multi-set convolutional network to encode tables, joins, and predicates. The outputs of the networks are averaged, concatenated, and fed to a final neural network for estimation.

ROBUST-MSCN [33] develops from MSCN by introducing query feature masks and join sample bitmaps to make the model could handle dynamic workload and data robustness.

ALECE [22] is a state-of-the-art method, which proposes an attention based methods for queries under dynamic workloads. It consists of a data-encoder to learn column correlations, and a query-analyzer to capture correlation between the outputs of data-encoder and the query cardinalities.

CostE aims to predict the execution cost (execution time) of a query plan. Learned cost estimators encode query plans as input features and predict execution costs. Several downstream tasks depend on cost estimation, such as join order selection and index selection [13, 41, 54]. We evaluate three representative cost estimation methods: two early methods QPP-NET [28] and END-TO-END [42], and one recent method QUERYFORMER [53]. We do not include database-agnostic methods [14, 23] since they learn from multiple datasets and workloads, which falls outside our experimental setting.

END-TO-END [42] employs a tree-structured neural network to encode features of query plans and database statistics, and aggregate them over the query graph to predict query plan latency.

QPP-NET [28] also incorporates plan structure using MLPs trained per operator type (e.g., hash join, nested loop join). Each unit processes operator-specific features to predict per-operator runtime

and hidden states, which are propagated through the query graph, and additionally leverages estimated cardinalities and operator costs.

QueryFormer [53] employs transformers to estimate query costs and introduces a tree-based self-attention mechanism to learn from long query plans with many operators. It also incorporates bitmap samples and histograms as input features.

QO aims to generate query plans and select the most efficient plan for a given query. While it relies on cost estimation, recent methods often adopt end-to-end frameworks with online learning. Therefore, we include query optimization in our evaluation. Learning-based query optimizers either assist the expert optimizer by suggesting per-query hints based on execution feedback, or learn to directly generate better query plans. We include both types of methods in our experiments to evaluate how workload drift affects their performance. We select three representative learned query optimizers: an earlier classic optimizer-augmented model Bao, a recent ranking based optimizer Lero, and a recent reinforcement learning based optimizer Balsa.

Bao [26] is a learned query optimization system that enhances the traditional optimizer by selecting per-query hints.

Lero [55] improves query optimization performance by learning ranking scores of query plans rather than predicting latency.

Balsa [49] is a deep reinforcement learning-based optimizer that learns to optimize queries without relying on an expert optimizer. Balsa first acquires basic knowledge from a simple, environment-agnostic simulator, followed by safe continuous learning in real executions.

Other database components, such as knob tuning, are largely dependent on physical system configurations rather than solely on workload features, and are thus excluded from this study.

We take the performance of PostgreSQL 13.1 as baseline for all database components, since it is open-source, widely-used in prior learned database component evaluations, and not affected by workload drift.

3.2 Evaluation Setup

Workloads. We evaluate the four types of workload drift using the two proposed benchmark workloads, JOB-light-drift and STATS-drift. The workload settings and generation procedures follow the methodology described in Section 2.3 and Table 4, where each workload contains 2,000 queries. For cardinality estimation, the two workloads used to evaluate output drift are partitioned based on the actual cardinalities of the generated queries. For cost estimation and query optimization, the two workloads used to evaluate output drift are partitioned based on the execution time of the generated queries.

Evaluation metrics. We focus on the evaluation metrics which are affected by workload drift.

Estimation accuracy (for CE and CostE). This metric evaluates whether the considered methods are robust to workload drift. We measure estimation accuracy using the q-error, defined as $\frac{\max(est,act)}{\min(est,act)}$, where est and act denote the estimated and actual cardinalities for CE, and the estimated and actual execution times for CostE, respectively.

End-to-end execution time (for CE, CostE and QO). This metric captures the impact of workload drift on overall query execution performance, reflecting whether degradation in estimation accuracy translates into slower query execution.

Training time (for QO). This metric measures the total training cost. Since Lero and Balsa follow a learn-to-optimize paradigm that requires executing queries in each training epoch, their training cost is highly sensitive to the training workload. In contrast, the training time for cardinality estimation and cost estimation models is not affected by workload drift, so we do not include it in the evaluation.

We do not evaluate other common metrics, such as query inference time and space cost, since these metrics are not affected by workload drift.

Model training and implementation. We follow the original papers and publicly released source code of all baselines [1–3, 9, 19, 30, 37–39], as well as a unified cost estimation evaluation platform [13, 17], to implement the models and tune hyper-parameters.

CE. For end-to-end query execution time evaluation, we extend PostgreSQL following prior work [12] to incorporate learned cardinality estimators into query execution. We employ an early stopping strategy: training terminates if the validation loss does not improve by at least 0.1% or when a maximum of 100 epochs is reached, whichever occurs first.

CostE. To evaluate end-to-end query execution time, we follow prior work that integrates learned cost estimators into Bao [26, 54], enabling PostgreSQL to utilize cost estimations produced by external models. We apply the same early stopping strategy as in CE, terminating training when the validation loss does not improve by at least 0.1% or after 100 epochs.

QO. Bao adopts an offline training strategy: query execution latencies are first collected by executing all queries in PostgreSQL, and the prediction model is then trained on these measurements. In contrast, Lero and Balsa follow a learn-to-optimize paradigm, generating and executing query plans at each training epoch. To ensure reasonable training time, each training workload contains 200 queries, and all models are trained for 10 epochs.

Unless otherwise stated, all experiments are repeated five times, and the reported results correspond to the mean values of the evaluation metrics.

All methods of selected database components and PostgreSQL are evaluated on a same machine with a Intel(R) Xeon(R) CPU E5-2698 v4 and 512 GB memory. All learned models strictly follow the setup described in the original papers, including the model training pipeline and evaluation protocol. For the learned query optimization methods and query execution evaluations using PostgreSQL, we adopt the same PostgreSQL configuration parameters as Balsa to ensure consistency with prior work, specifically setting *shared_buffers* to 8GB and *max_worker_processes* to 16.

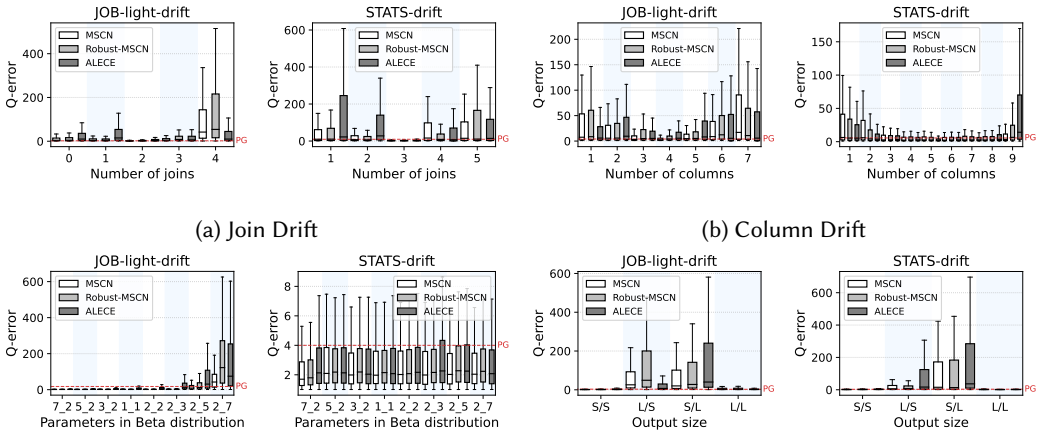
4 Cardinality Estimation

In this section, we evaluate the impact of workload drift on cardinality estimation. We begin with performance analysis. Then we present the key findings and insights derived from the evaluation.

4.1 Analysis of Performance

Figure 4 and Figure 5 illustrate the estimation performance and end-to-end query execution time under different workload drifts to answer **RQ2** and **RQ3** in Section 1. We have the following observations.

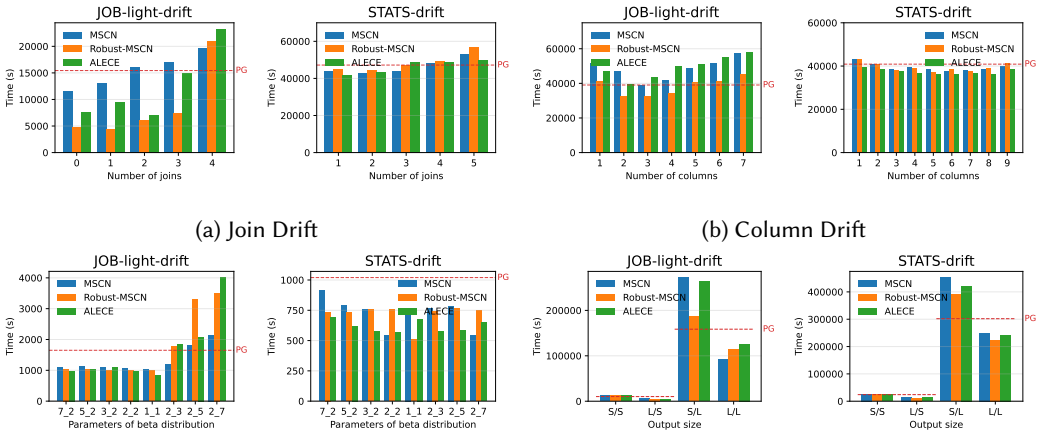
O1. Join Drift substantially reduces estimation accuracy, but its impact on query execution performance is asymmetric. As shown in Figure 4a, when training and testing workloads share the same number of joins (2 joins for JOB-light-drift and 3 joins for STATS-drift), the median error of all estimators remains relatively low at 1.7—2.8. In contrast, under Join Drift, the median error rises to 4.0–30, representing up to a 10× increase compared to the no-drift setting. Moreover, the estimation error grows as the difference in join counts between training and testing workloads increases. When the degree of drift is sufficiently large, the learned estimators perform worse than PostgreSQL. We further observe differences among models in their robustness to Join Drift: ALECE is most sensitive, followed by MSCN, while Robust-MSCN demonstrates the highest resilience. This robustness of MSCN and Robust-MSCN can be attributed to their use of sample bitmap dataset statistics, and in particular, Robust-MSCN leverages feature masks to mitigate the impact of workload drift.



(a) Join Drift (b) Column Drift

(c) Value Drift (d) Output Drift

Fig. 4. Q-error of cardinality estimation. The red dashed line is the median q-error of PostgreSQL.



(a) Join Drift (b) Column Drift

(c) Value Drift (d) Output Drift

Fig. 5. Query execution time of cardinality estimation

Interestingly, the decline in estimation accuracy does not always translate into worse query execution performance as demonstrated in Figure 5a.When the training workload contains queries with fewer joined tables, execution performance is often better than when the training workload contains queries with the same or a larger number of joins. This phenomenon arises because the traditional query optimizers in DBMS relies on accurate cardinality estimations for sub-queries to determine the overall query plan cost. Estimators trained on smaller-join queries can often provide more accurate sub-query cardinalities, thereby enabling the optimizer to select better query plans, even if their overall estimation accuracy appears degraded under Join Drift.

When errors become too large (e.g., 4 joins on JOB-LIGHT-DRIFT), the estimation no longer meaningfully guides join-order selection or operator choice. The optimizer effectively behaves as if it were receiving non-informative estimations. As a result, execution-time differences are

dominated by physical operator behavior rather than by the relative accuracy of different CE models, and thus the trends in q-error do not correspond to the trends in execution time.

O2. Column Drift moderately reduces estimation accuracy and slightly affects query execution performance. Figure 4b shows that when training and testing workloads have the same number of filter columns (4 columns for JOB-light-drift and 5 columns for STATS-drift), the median error of all estimators is relatively low, ranging from 2.24—5.56 on JOB-light-drift and 1.91—2.09 on STATS-drift. As the drift enlarges, the error increases notably, reaching 11.2—57.5 on JOB-light-drift and 4.83—5.41 on STATS-drift, which exceed the error observed for PostgreSQL under large workload drift. Similar to Join Drift, ALECE suffers the most, while MSCN and Robust-MSCN achieve better robustness due to their incorporation of dataset statistics during training.

The impact on query execution time is much smaller than the impact on estimation accuracy as demonstrated in Figure 5b. Under Column Drift, the end-to-end query execution time increases only slightly about 1.08—1.16× compared to the no-drift case. This moderate effect arises because (1) the optimizer may still choose the same query plan even when cardinality estimations are inaccurate, and (2) even when different plans are selected, their execution times can be similar. As a result, the degradation in estimation accuracy does not necessarily lead to significant slowdowns in query execution under Column Drift.

O3. Value Drift impacts estimation accuracy and end-to-end execution performance consistently, with dataset-specific differences. Figure 4c shows that the effect of Value Drift on estimation accuracy varies depending on the dataset and the underlying column distributions. On JOB-light-drift, estimation accuracy remains relatively stable under small drifts (from $(7, 2)$ to $(2, 3)$), but degrades significantly under larger drifts (e.g., $(2, 5)$ and $(2, 7)$), eventually becoming worse than PostgreSQL under large drift. In contrast, on STATS-drift, estimation accuracy remains unchanged across different value distributions in the training workloads. This discrepancy arises because the effect of Value Drift depends on the distribution of the filtered columns. Table 5 reports the Kolmogorov–Smirnov distance (KS) for the filtered columns, where a lower value indicates a distribution closer to uniform. Although STATS exhibits high skewness overall, the columns involved in Value Drift do not necessarily follow the same pattern. For instance, two STATS tables (p and u) contain predicate columns dominated by a single frequent value, resulting in a large KS distance but minimal impact on cardinality estimation. Therefore, the skewness relevant to Value Drift is not higher for STATS than for IMDb. This emphasizes that for Value Drift, it is not the overall dataset skewness, but the skewness of the specific predicate columns that matters.

Table 5. Skewness of columns used in Value Drift

Dataset	IMDb			STATS			
Table	ci	mi	t	b	pH	u	v
KS	0.46	0.76	0.49	0.21	0.83	0.99	0.28

when the underlying column distribution is nearly uniform, Value Drift has little to no impact, whereas when the distribution is skewed, Value Drift leads to degradation of estimation accuracy.

The trend in query execution time mirrors that of estimation accuracy as illustrated in Figure 5c. These findings demonstrate that the impact of value drift on query execution is consistent with its impact on estimation accuracy.

O4. Output Drift reduces estimation accuracy significantly, but training on workloads with large cardinalities always have better end-to-end query execution performance.

Figure 4d demonstrates that the median error on the case with Output Drift could be 5–37× to the case without Output Drift. The estimation error on STATS-DRIFT is significantly higher than that on JOB-LIGHT-DRIFT, as STATS-DRIFT exhibits stronger column correlations and higher skewness, as demonstrated in Table 3. However, Figure 5d shows that training on workloads with large cardinalities always has much better query execution performance than training on workloads with small cardinalities.

4.2 Key Findings and Insights.

We summarize the key findings for cardinality estimation as follows:

- **Impact of workload drift on estimation accuracy:** The impact of different types o f workload drift on estimation accuracy follows the ranking: Output Drift > Join Drift ≫ Column Drift > Value Drift. For Output Drift and Join Drift, even small amounts of drift can significantly degrade estimation accuracy. In contrast, for Column Drift and Value Drift, only large drift levels result in noticeable accuracy degradation.
- **Impact of workload drift on end-to-end query execution time:** The impact on query execution time follows a similar ranking: Output Drift > Join Drift > Column Drift > Value Drift. Output Drift leads to a significant increase in query execution time. For Join Drift, Column Drift, and Value Drift, small levels of drift have a minor effect on execution time, but large levels of drift cause a significant increase in query execution time.
- **Discrepancy between estimation accuracy and query execution time:** The relationship between workload drift and end-to-end query execution time does not fully align with its impact on estimation accuracy. For both Value Drift and Output Drift, the decrease in estimation accuracy directly correlates with an increase in end-to-end query execution time as the degree of drift increases. However, for Column Drift, although both estimation accuracy and execution time increase with drift, the impact on estimation accuracy is more pronounced. For Join Drift, the effects on estimation accuracy and query execution time differ with the drift level: training on workloads with fewer joins reduces estimation accuracy but improves query execution performance.
- **Workload drift types to prioritize in future work:** Future research should focus primarily on Output Drift, as it has the largest impact on both estimation and query optimization performance. For Column Drift and Value Drift, investigations should emphasize large-drift scenarios, since small drift generally has minor effects on the performance. Regarding Join Drift, estimators should be trained on queries with a small number of joins. Although Join Drift can significantly reduce estimation accuracy, query optimization relies on cardinality estimations for sub-queries, which typically involve fewer joins.

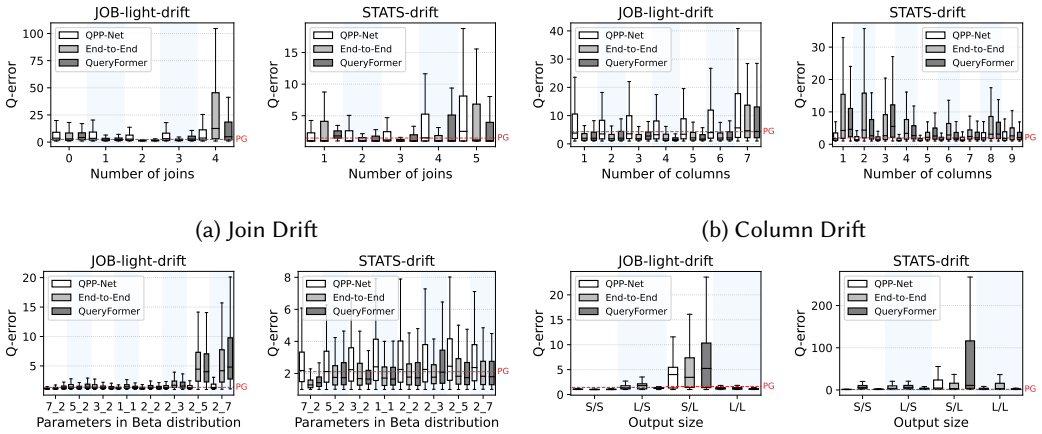
5 Cost Estimation

In this section, we assess how workload drift affects cost estimation. We begin with a detailed performance analysis. Then we highlight the key findings and insights derived from the evaluation.

5.1 Performance Analysis

Figure 6 and Figures 7 demonstrate the cost estimation accuracy and end-to-end query execution time under different types and levels of workload drift to answer **RQ2** and **RQ3** in Section 1. We have the following observations.

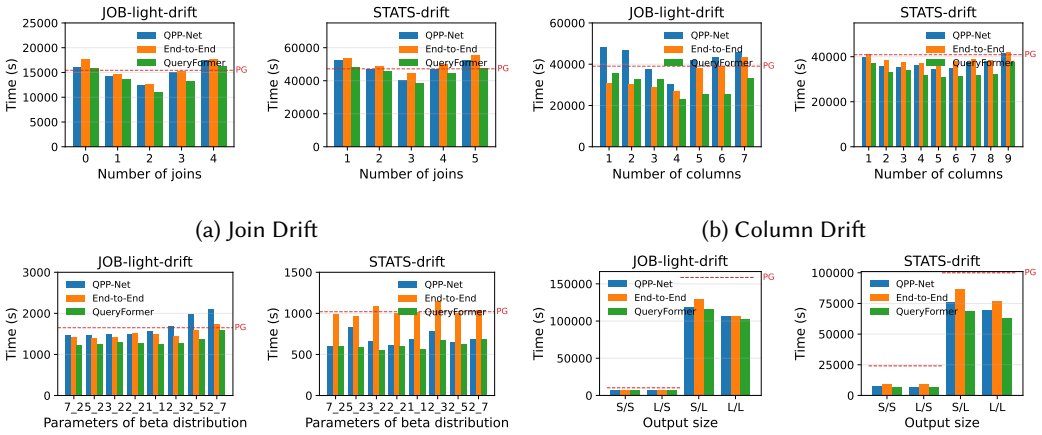
1. Increasing Join Drift significantly reduces estimation accuracy, but only moderately affects end-to-end query execution time. Figure 6a shows that higher levels of Join Drift substantially degrade estimation accuracy, with the median error increasing by up to 4.5× on



(a) Join Drift (b) Column Drift

(c) Value Drift (d) Output Drift

Fig. 6. Q-error of cost estimation. The red dashed line is the median q-error of PostgreSQL.



(a) Join Drift (b) Column Drift

(c) Value Drift (d) Output Drift

Fig. 7. Query execution time of cost estimation

STATS-DRIFT and 1.5× on JOB-LIGHT-DRIFT. As the drift increases, the performance of the learned models gradually deteriorates and eventually becomes worse than that of PostgreSQL. The effect is more pronounced on STATS-DRIFT due to its higher data skewness and stronger inter-column correlations.

However, the impact of Join Drift on end-to-end query execution time is moderate. The execution time under Join Drift increases by at most 1.44× compared to the no-drift case. This observation can be attributed to the non-monotonic relationship between cost estimation accuracy and query optimization performance: inaccurate cost estimates may still lead the optimizer to select near-optimal plans, whereas accurate estimates do not necessarily guarantee the optimal plan selection.

O2. Increasing Column Drift moderately reduces estimation accuracy and end-to-end query execution performance. Figure 6b illustrates cost estimation accuracy under varying levels of Column Drift. The median error remains largely unchanged as the Column Drift increases,

indicating that the number of predicate columns has a minor effect on query execution cost. However, the error distribution exhibits a pronounced long tail with occasional extreme outliers. In particular, under large Column Drift, the maximum error reaches up to 400× on JOB-LIGHT-DRIFT and 100× on STATS-DRIFT, substantially exceeding the errors observed for PostgreSQL. This occurs because queries with more predicate columns often have smaller actual costs, as the additional predicates reduce the number of tuples satisfying the query.

Column Drift also increases end-to-end query execution time. The query execution time under Column Drift is at most 1.20× that of the no-drift case on JOB-LIGHT-DRIFT and 1.53× on STATS-DRIFT. These results indicate that Column Drift has non-negligible impact on query optimization performance.

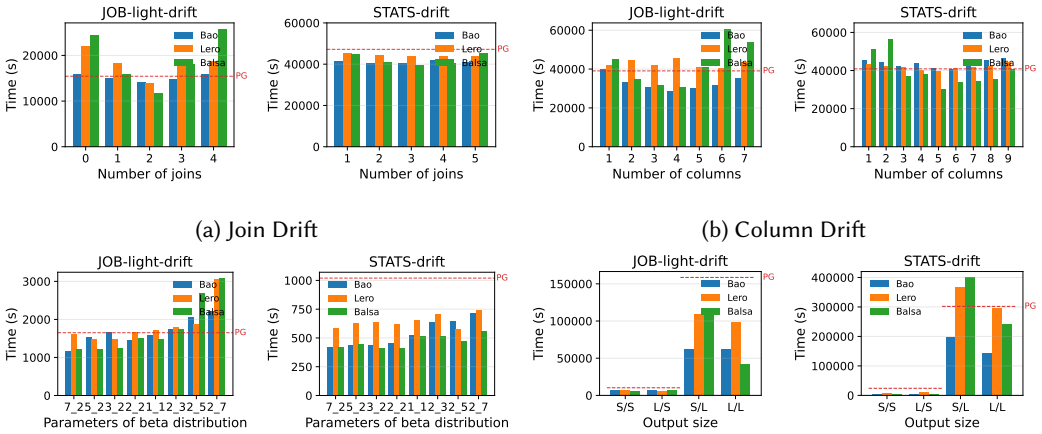
O3. Large Value Drift slightly reduces cost estimation accuracy and end-to-end query execution performance. Figure 6c shows that when Value Drift is small (from Beta$(7, 2)$ to Beta$(3, 2)$), both cost estimation accuracy and end-to-end query execution time remain largely unchanged. As the drift level increases further (from Beta$(2, 2)$ to Beta$(2, 7)$), the median estimation error rises up to 1.42× compared to the no-drift case, and the end-to-end execution time increases up to 1.27×. These results indicate that only large Value Drift has a noticeable impact on cost estimation performance.

O4. Output Drift significantly reduces cost estimation accuracy but has minimal effect on end-to-end query execution time. Figure 6d illustrates cost estimation accuracy under Output Drift. On JOB-LIGHT-DRIFT, the median error increases up to 1.35× compared to the no-drift case, while on STATS-DRIFT, it rises up to 3.13×. Output Drift has a stronger impact on STATS-DRIFT due to its more complex query plan structures and higher query costs. In contrast, end-to-end query execution time is only slightly affected. As shown in Figure 7d, execution time under Output Drift increases by at most 1.19× compared to the no-drift case. This indicates that reductions in estimation accuracy caused by Output Drift do not necessarily translate into substantial degradation in query optimization performance.

5.2 Key Findings and Insights

We summarize the key findings for cost estimation as follows:

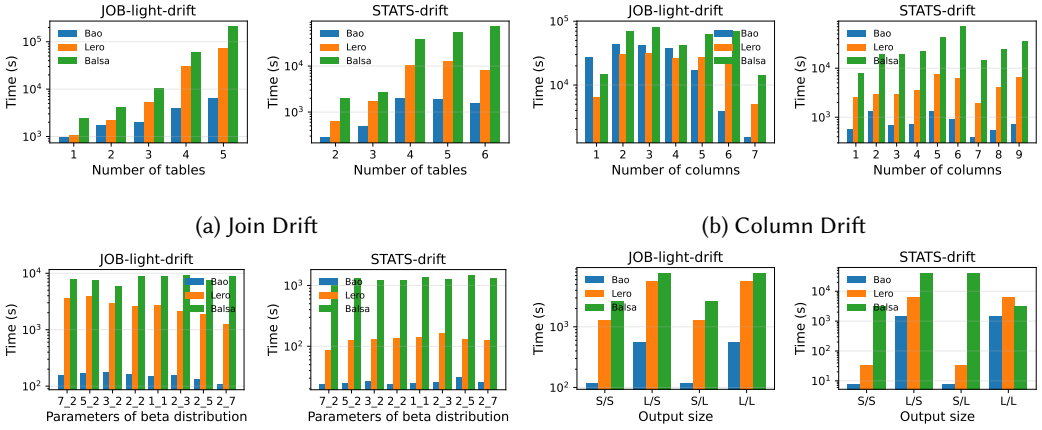
- **Impact of workload drift on estimation accuracy:** The impact of different types of workload drift on cost estimation accuracy follows the ranking: Join Drift ≈ Output Drift > Column Drift > Value Drift. For Join Drift, even small amounts of drift can substantially degrade estimation accuracy. Output Drift also introduce significant estimation accuracy degradation, especially on workloads with high column correlations (e.g., STATS-DRIFT). Column Drift only generate large error on extreme cases, and Value Drift has almost no effect on estimation accuracy.
- **Impact of workload drift on end-to-end query execution time:** The impact on end-to-end query execution time follows a slightly different ranking: Join Drift > Column Drift > Value Drift > Output Drift. Join Drift causes the most pronounced increase in execution time. Column Drift and Value Drift slightly increase execution time under small drift, whereas moderately increase query execution time under large drift. Output Drift can only slightly increase query execution time.
- **Discrepancy between estimation accuracy and end-to-end execution:** The relationship between cost estimation accuracy and query execution time does not fully align for all drift types. Output Drift has large impact on estimation accuracy but minimal impact on query execution time. For Join Drift, Column Drift and Value Drift, even if increasing the level of drift could reduce estimation accuracy and increase query execution time, the level of increasing query execution time is smaller than the level of decreasing estimation accuracy.



(a) Join Drift (b) Column Drift

(c) Value Drift (d) Output Drift

Fig. 8. Query execution time of query optimization



(a) Join Drift (b) Column Drift

(c) Value Drift (d) Output Drift

Fig. 9. Training time of query optimization

- **Workload drift types to prioritize in future work:** Future research should focus primarily on Join Drift, as it has the strongest impact on both cost estimation accuracy and query optimization performance. For Column Drift and Value Drift, investigations should emphasize large-drift scenarios, since small levels of drift generally have negligible effects on estimation and optimization. With respect to Output Drift, attention should be given to queries with large execution costs, as estimation and optimization for small-cost queries typically remain robust.

6 Query Optimization

This section explores the impact of workload drift on query optimization. We begin with a comprehensive performance evaluation. Then we provide the key insights and findings based on the results.

6.1 Analysis of Performance

Figure 8 and Figure 9 illustrate the end-to-end query execution time and training cost under different types and levels of workload drift, aiming to answer **RQ3** in Section 1. Here, we focus on how each optimizer behaves under varying workload drift rather than comparing the overall performance across optimizers. Since Bao employs offline learning while Lero and Balsa adopt "learn-to-optimize" strategy, the training times of Lero and Balsa are significantly higher than that of Bao. Consequently, our analysis focuses primarily on the training times of Lero and Balsa. From these experiments, we make the following observation:

O1. Training queries with fewer joins than the testing workload can reduce performance, while training on queries with more joins substantially increases training cost. As shown in Figure 8a, the end-to-end query execution time remains largely unchanged in most cases. However, when the number of joins in the training workload is smaller than that in the testing workload, query optimization performance degrades. This is because queries with fewer joins do not provide sufficient training examples to cover the space of possible query plans for the target queries. Bao is robust to Join Drift and keeps good performance under large Join Drift, since it utilize the query plans generated by Postgres and only provide hints to the query optimizers in Postgres.

Conversely, Figure 9a demonstrates that training on workloads with a larger number of joins dramatically increases the training cost. Specifically, training on queries with more than 3 joins requires over 10× the time compared to workloads with 1—2 joins. The increase arises from the higher complexity of multi-join queries, which take longer to execute and to process during model training.

These results suggest a practical trade-off: training on queries with the same or slightly fewer joins than the target workload can reduce training cost while maintaining good performance.

O2. End-to-end performance is slightly affected by column drift, while training time varies across workloads with different numbers of filter columns. Figure 8b illustrates the end-to-end query execution time under varying levels of Column Drift. As the Column Drift increases, execution time rises moderately, reaching up to 1.06× on Bao, 1.12× on Lero, and 1.48× on Balsa. Bao maintains stable performance under Column Drift because it only provides hints to PostgreSQL rather than generating its own query plans. Lero predicts ranking scores rather than absolute query latency, which makes it more robust than Balsa. We also observe that Column Drift has a greater impact on STATS-drift than on JOB-light-drift, due to higher column skewness and stronger correlations in the STATS dataset compared to IMDb.

Figure 9b shows that training time does not exhibit a clear pattern with respect to Column Drift. This is because training time is primarily determined by the join graph of each query, which is randomly selected in the training workloads, and thus independent of the number of filter columns.

O3. Only large value drift slightly affects end-to-end query execution time, while training time remains unaffected. Figure 8c illustrates that when the Value Drift is small (from $(7, 2)$ to $(2, 2)$), the end-to-end query execution time remains largely unchanged. As the level of Value Drift increases further, execution time increases modestly, reaching up to 1.32× compared to the no-drift case. We also observe that large Value Drift has a greater impact on JOB-light-drift than on STATS-drift, despite the STATS dataset exhibiting higher column skewness and stronger column correlations. This is because end-to-end query execution time is primarily determined by the query templates rather than the sampled predicate values. In STATS-drift, the query template used for Value Drift consistently generates similar join orders, indicating that the predicate value distribution has little impact on join order selection. Hence, Value Drift has a minimal effect on STATS-drift.

Training time remains almost identical across different levels of Value Drift, since queries with the same template typically exhibit similar execution latency.

O4. Output drift has larger impact on STATS-DRIFT than on JOB-LIGHT-DRIFT. Figure 9d shows that large Output Drift can significantly degrade query optimization performance. On STATS-DRIFT, the end-to-end execution time increases up to 1.47× compared to the no-drift case, whereas on JOB-LIGHT-DRIFT, the increase is only about 1.10×. The greater sensitivity of STATS-DRIFT to Value Drift arises from higher column skewness and stronger inter-column correlations, which result in larger variations in query execution characteristics, making query optimization more challenging.

Moreover, training cost is highly dependent on query execution latency (i.e., the output). Training on workloads with large outputs can require more than 100× the time compared to workloads with small outputs, reflecting the additional computational burden imposed by high-latency queries.

6.2 Key Findings and Insights

We summarize the key findings for query optimization as follows:

- **Impact of workload drift on end-to-end query execution time:** The impact ranking of workload drift types on execution time is: Output Drift > Join Drift > Column Drift > Value Drift. Specifically, large Output Drift can significantly increase query execution time, particularly on STATS-DRIFT, where end-to-end time increases up to 1.47× under large Output Drift, while the increase is only about 1.10× on JOB-LIGHT-DRIFT. For Join Drift, Column Drift, and Value Drift, small drift has minor effects, but large drift can still noticeably increase execution time. This demonstrates that the dataset characteristics, such as column skewness and correlations, influence how workload drift translates into query execution performance degradation.
- **Impact of workload drift on training time:** Training cost is highly sensitive to the characteristics of the workload. For Join Drift, training on queries with more joins dramatically increases training time (over 10× for queries with more than 3 joins compared to 1—2 joins). For Output Drift, training on workloads with large label values can require more than 100× the time compared to workloads with small labels, reflecting the additional computational burden imposed by high-latency queries. Column Drift and Value Drift have comparatively moderate effects on training time.
- **Dataset-dependent effects:** The impact of workload drift on end-to-end execution time is dataset-specific. STATS-DRIFT is more sensitive to label and Value Drift due to higher column skewness and stronger correlations, leading to larger deviations in query execution plans and runtime. In contrast, JOB-LIGHT-DRIFT shows smaller increases in execution time under the same drift levels, reflecting more uniform data distributions and less complex correlations.
- **Workload drift types to prioritize in future work:** For online learned query optimization, future work should prioritize training on queries with a small number of joins while testing on queries with larger numbers of joins, as training time grows significantly with the number of joins. Research should also focus on robustness or consider retraining under large Column Drift and Value Drift, since small levels of these drifts have minimal impact on performance. Additionally, training on queries with smaller execution latencies can substantially reduce training time without compromising query optimization performance.

7 Related Work

Workload-driven learned database components. Recent approaches leveraging learning-based techniques for various database components have demonstrated substantial performance improvements over traditional methods. Many of these methods utilize query workloads to train models,

including workload-driven cardinality estimation [15, 32, 36, 52], cost estimation [24, 29, 42, 53], query optimization [26, 27, 49, 55], index selection [7, 31, 41], and knob tuning [21, 51]. In this work, we focus specifically on cardinality estimation, cost estimation, and query optimization.

Cardinality estimation. Cardinality estimation predicts the result size of a query or subquery. Learned estimators are often divided into data-driven and workload-driven approaches. Since data-driven methods mainly learn the data distribution and are typically less sensitive to workload variation, we focus on workload-driven estimators that encode query structures and predicates [15, 22, 24, 33, 36, 45, 52]. NN [8] maps predicate features to cardinalities with multi-layer networks. MSCN [15] aggregates set-encoded tables, joins and predicates for cardinality prediction, while Robust-MSCN [33] improves robustness via dropout and feature masking. QuickSel [36] adopts workload-driven Gaussian mixture models, and ALECE [22] leverages attention-based query encoding with correlation modeling to improve estimation quality.

Cost estimation. Cost estimation predicts the execution cost or latency of a query plan. Learned cost models typically take physical plans and database statistics as input and output runtime estimates. Flat-Vector [10] uses fixed-dimensional operator features with regression, End-to-End [42] encodes queries and physical operators as features and embeds them into a tree-structured model to predict query latency. QPP-Net [28] introduces neural units that produce latency estimates and performance-related properties, dynamically assembling operator-level neural units into a neural network to predict query plan latency. QueryFormer [53] further employs transformer-style self-attention to better handle long plans. Database-agnostic models (e.g., Zero-shot [14] and DACE [23]) require training across multiple datasets and workloads, and are thus outside our experimental scope.

Query optimization. Query optimization explores the plan space and selects the lowest-cost plan [4, 5, 26, 27, 49, 55]. Learning-based optimizers broadly fall into two categories. The first class generates query plans using model-based approaches with cost estimation. DQ [16] and Neo [27] learn from cost models from expert optimizers. Balsa [49] starts from a optimizer generated cost model and improves through iterative execution feedback. The second class leverages traditional query optimizers to generate candidate plans and uses learning-based models to select the optimal plan. Bao [26] learns plan representations for selection, HybridQO [50] combines cost-based and learning-based signals, and Lero [55] uses learning-to-rank plans instead of directly regressing latency.

Workload drift problem. A key assumption for learned workload-driven database components to maintain high performance is that the training queries follow the same distribution as the testing queries. When the distributions differ, the performance of these models can degrade significantly [20, 47]. Retraining the model to accommodate the new workload is often costly, as it requires query execution in the DBMS, which is time-consuming.

Existing solutions to workload drift. Warper [20] and ShiftHandler [47] are not learned models. Their goal is to reduce retraining cost by selecting representative queries once workload drift is detected. Both methods assume that the learned model must be retrained and focus on identifying a small but informative subset of the target workload for retraining. Specifically, Warper maintains a representative query pool and uses a GAN [11] to generate representative queries when a new workload arrives. ShiftHandler adopts the k-medoids method to select representative queries from the new workload for retraining. In this sense, they are add-on procedures that can be combined with any learned models. In contrast, our work studies how learned models behave under workload drift without retraining. Therefore, the problems addressed by Warper and ShiftHandler are orthogonal to the problem studied in this paper. Robust-MSCN [33] addresses drift by introducing feature masking and dropout into MSCN, and we include this method in our evaluation. Zero-shot [14] and DACE [23] aim to build database-agnostic cost estimation models and require training across

Table 6. Key cross-component findings under workload drift

Finding	CE	CostE	QO
Dominant drift type	Join Drift / Output Drift		
Fewer joins on estimation accuracy	↓	↓	N.A.
Fewer joins on execution performance	↑	↓	↓
Output drift on estimation accuracy	↓	↓	N.A.
Output drift on execution performance	↓	≈	↓

multiple datasets and workloads. This setting requires substantially larger training workloads and training time [13], which is fundamentally different from our experimental setting. Therefore, we do not include them in our experiments.

Adaptive query processing techniques in RDBMS. Some widely-used RDBMS provide adaptive query-processing techniques. SQL Server implements batch-mode adaptive joins, which dynamically switch between hash and nested-loops joins at runtime, and adjusts cardinality estimation based on past executions. Oracle supports dynamic plan re-optimization, allowing the engine to revise execution plans when runtime statistics deviate significantly from expectations. These mechanisms can improve performance for structurally similar queries, but they do not address systematic workload drift involving new join structures and shifted predicate distributions. Hence, these engines remain vulnerable to workload drift.

8 Conclusions and Future Directions

8.1 Conclusions and Key Takeaways

We summarize four key findings to answer **RQ1** to **RQ4** in Section 1.

Robust workload-driven database component design should prioritize addressing Join Drift, Output Drift, and large Column Drift or Value Drift. Our evaluations across all three database tasks demonstrate that Join Drift has the most substantial impact on performance: even small join drift can significantly reduce model accuracy and moderately increase end-to-end query execution time. Output Drift similarly degrades model accuracy and increases query execution time. In contrast, Column Drift and Value Drift have only moderate effects, and their impact becomes noticeable only under large drift levels.

Workload-driven component design should prioritize end-to-end query execution performance rather than model-level accuracy. The impact of workload drift on model estimation accuracy does not always align with its effect on actual query execution performance. A degradation in estimation accuracy under workload drift does not necessarily translate into a deterioration in query execution performance as summarized in Table 6. For instance, in cardinality estimation, training on queries with fewer joins reduces accuracy, yet improve query optimization performance. Similarly, in cost estimation, Output Drift can lead to substantial deterioration in estimation accuracy while having only a marginal impact on query optimization. These observations indicate that estimation accuracy and execution performance are not always tightly coupled, and their sensitivity to workload drift can differ across tasks. Therefore, the design of workload-driven database components should prioritize optimizing end-to-end query performance rather than solely minimizing estimation error.

Learn-to-optimize database components should focus on improving generalization from limited or biased training workloads for evolving workloads. Unlike one-shot learned models that are trained once on large, static workloads, learn-to-optimize components execute queries during each training iteration to collect feedback for model updates. To reduce training overhead,

they are typically trained with a small number of queries, which leads to limited or biased training samples. Hence, their robustness under workload drift depends on the ability to generalize from sparse or skewed training data. Therefore, the design of learn-to-optimize database components should emphasize developing mechanisms that improve generalization and stability across evolving workloads.

Workload-driven components should more focus on datasets with strong correlations and high skewness. Our experiments demonstrate that datasets exhibiting strong correlations and high skewness are more sensitive to workload drift, especially on Join Drift and Column Drift, as changes in the structural relationships disturb the underlying patterns upon which workload-driven components depend, which results in greater model performance degradation. Therefore, when designing and evaluating workload-driven components, it is crucial to account for these data characteristics, as they directly influence the robustness of models under varying drift conditions. The trend of query execution time does not always match the estimation accuracy because execution performance is influenced by additional factors such as the dominant join operators. For example, execution time is dominated by heavy physical operators in STATS, which reduces the relative impact of drift-induced estimation errors, leading to smaller changes in runtime.

8.2 Future Directions

Can we design training workload with minimal number of queries while keeping good performance of database components? Most workload-driven database components assume there are enough query workloads for training models. However, training data with ground-truth can only be obtained through actual query execution in the DBMS, which is a costly process. Therefore, designing sufficient yet minimal training workloads while keeping good model performance that reduce the query execution cost required to obtain ground-truth outputs is an interesting direction.

Can we design robust database components with small training data? In real-world scenarios, it is often difficult to obtain sufficient or comprehensive query workloads for training. Models trained on such limited data are more likely to suffer from workload drift. Designing robust models that can generalize well under insufficient training workloads remains an open and largely unexplored problem.

How to perform fast cold-start for online learning-based database components? Online learning-based database components, such as learned query optimizers, require training during query execution rather than relying on historical workloads. This process is time-consuming and delays model readiness. Therefore, accelerating the cold-start phase for online learning models is a promising research direction.

Acknowledgments

This research is supported in part by Singapore MOE AcRF Tier-2 Grant (T2EP20122-0003, T2EP20224-0005, T2EP20223-0004).

References

[1] ALECE code 2023. *Source code of ALECE.* https://github.com/pfl-cs/ALECE
[2] Balsa code 2022. *Source code of Balsa.* https://github.com/balsa-project/balsa
[3] Bao code 2020. *Source code of Bao.* https://github.com/learnedsystems/BaoForPostgreSQL/tree/master
[4] Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. 2023. Loger: A learned optimizer towards generating efficient and robust query execution plans. *Proceedings of the VLDB Endowment* 16, 7 (2023), 1777–1789.
[5] Xu Chen, Haitian Chen, Zibo Liang, Shuncheng Liu, Jinghong Wang, Kai Zeng, Han Su, and Kai Zheng. 2023. Leon: A new framework for ml-aided query optimization. *Proceedings of the VLDB Endowment* 16, 9 (2023), 2261–2273.

[6] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek Narasayya. 2021. DSB: A decision support benchmark for workload-driven and traditional database systems. *Proceedings of the VLDB Endowment* 14, 13 (2021), 3376–3388.

[7] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R Narasayya. 2019. Ai meets ai: Leveraging query executions to improve index recommendations. In *Proceedings of the 2019 International Conference on Management of Data*. 1241–1258.

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

[9] End-to-End code 2019. *Source code of End-to-End.* https://github.com/greatji/Learning-based-cost-estimator

[10] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L Wiener, Armando Fox, Michael Jordan, and David Patterson. 2009. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 592–603.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27.

[12] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, et al. 2021. Cardinality estimation in DBMS: a comprehensive benchmark evaluation. *Proceedings of the VLDB Endowment* 15, 4 (2021), 752–765.

[13] Roman Heinrich, Manisha Luthra, Johannes Wehrstein, Harald Kornmayer, and Carsten Binnig. 2025. How good are learned cost models, really? Insights from query optimization tasks. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–27.

[14] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-shot cost models for out-of-the-box learned cost prediction. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2361–2374.

[15] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677* (2018).

[16] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2018. Learning to optimize join queries with deep reinforcement learning. *arXiv preprint arXiv:1808.03196* (2018).

[17] LCM experiment code 2025. *Source code of LCM.* https://github.com/DataManagementLab/lcm-eval/tree/main

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

[19] Lero code 2022. *Source code of Lero.* https://github.com/AlibabaIncubator/Lero-on-PostgreSQL/tree/main

[20] 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*. 1920–1933.

[21] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.

[22] Pengfei Li, Wenqing Wei, Rong Zhu, Bolin Ding, Jingren Zhou, and Hua Lu. 2023. ALECE: An Attention-based Learned Cardinality Estimator for SPJ Queries on Dynamic Workloads. *Proceedings of the VLDB Endowment* 17, 2 (2023), 197–210.

[23] Zibo Liang, Xu Chen, Yuyang Xia, Runfan Ye, Haitian Chen, Jiandong Xie, and Kai Zheng. 2024. DACE: A database-agnostic cost estimator. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4925–4937.

[24] Shuncheng Liu, Xu Chen, Yan Zhao, Jin Chen, Rui Zhou, and Kai Zheng. 2022. Efficient learning with pseudo labels for query cost estimation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 1309–1318.

[25] Qingzhi Ma and Peter Triantafillou. 2019. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proceedings of the 2019 International Conference on Management of Data*. 1553–1570.

[26] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *Proceedings of the 2021 International Conference on Management of Data*. 1275–1288.

[27] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: a learned query optimizer. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1705–1718.

[28] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-structured deep neural network models for query performance prediction. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1733–1746.

[29] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-structured deep neural network models for query performance prediction. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1733–1746.

[30] MSCN code 2019. *Source code of MSCN.* https://github.com/andreaskipf/learnedcardinalities/tree/master

[31] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning multi-dimensional indexes. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 985–1000.

[32] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-loss: learning cardinality estimates that matter. *Proceedings of the VLDB Endowment* 14, 11 (jul 2021), 2019–2032.
[33] Parimarjan Negi, Ziniu Wu, Andreas Kipf, Nesime Tatbul, Ryan Marcus, Sam Madden, Tim Kraska, and Mohammad Alizadeh. 2023. Robust query driven cardinality estimation under changing workloads. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1520–1533.
[34] NY Taxi 2025. *NY Taxi.* https://www.nyc.gov/site/tlc/about/about-tlc.page
[35] NYC flight 2025. *NYC flight.* https://www.kaggle.com/datasets/lampubhutia/nyc-flight-delay/data
[36] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. 2020. Quicksel: Quick selectivity learning with mixture models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1017–1033.
[37] QPP-Net code 2020. *Source code of QPP-Net.* https://github.com/rabbit721/QPPNet
[38] QueryFormer code 2023. *Source code of QueryFormer.* https://github.com/zhaoyue-ntu/QueryFormer
[39] Robust-MSCN code 2022. *Source code of Robust-MSCN.* https://github.com/learnedsystems/CEB
[40] Bart Samwel, John Cieslewicz, Ben Handy, Jason Govig, Petros Venetis, Chanjun Yang, Keith Peters, Jeff Shute, Daniel Tenedorio, Himani Apte, et al. 2018. F1 query: Declarative querying at scale. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1835–1848.
[41] Jiachen Shi, Gao Cong, and Xiao-Li Li. 2022. Learned index benefits: Machine learning based index performance estimation. *Proceedings of the VLDB Endowment* 15, 13 (2022), 3950–3962.
[42] Ji Sun and Guoliang Li. 2019. An end-to-end learning-based cost estimator. *Proceedings of the VLDB Endowment* 13, 3 (nov 2019), 307–319.
[43] US Census 2025. *US Census.* https://data.census.gov/
[44] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building an elastic query engine on disaggregated storage. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 449–462.
[45] Fang Wang, Xiao Yan, Man Lung Yiu, Shuai Li, Zunyao Mao, and Bo Tang. 2023. Speeding up end-to-end query execution via learning-based progressive cardinality estimation. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.
[46] Qiang Wang, Yi Shen, and Jian Qiu Zhang. 2005. A nonlinear correlation measure for multivariable data set. *Physica D: Nonlinear Phenomena* 200, 3-4 (2005), 287–295.
[47] Peizhi Wu and Zachary G Ives. 2024. Modeling Shifting Workloads for Learned Database Systems. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–27.
[48] Jingyi Yang, Peizhi Wu, Gao Cong, Tieying Zhang, and Xiao He. 2022. SAM: Database generation from query workloads with supervised autoregressive models. In *Proceedings of the 2022 International Conference on Management of Data*. 1542–1555.
[49] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a query optimizer without expert demonstrations. In *Proceedings of the 2022 International Conference on Management of Data*. 931–944.
[50] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. 2022. Cost-based or learning-based? A hybrid query optimizer for query plan selection. *Proceedings of the VLDB Endowment* 15, 13 (2022), 3924–3936.
[51] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 international conference on management of data*. 415–432.
[52] Kangfei Zhao, Jeffrey Xu Yu, Zongyan He, Rui Li, and Hao Zhang. 2022. Lightweight and accurate cardinality estimation by neural network gaussian process. In *Proceedings of the 2022 International Conference on Management of Data*. 973–987.
[53] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. Queryformer: A tree transformer model for query plan representation. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1658–1670.
[54] Yue Zhao, Zhaodonghui Li, and Gao Cong. 2023. A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. *Proceedings of the VLDB Endowment* 17, 4 (2023), 823–835.
[55] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A learning-to-rank query optimizer. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1466–1479.
[56] Daniel Zwillinger and Stephen Kokoska. 1999. *CRC standard probability and statistics tables and formulae.* Crc Press.

Received October 2025; revised January 2026; accepted February 2026