Modeling Shifting Workloads for Learned Database Systems

PEIZHI WU, University of Pennsylvania, USA
ZACHARY G. IVES, University of Pennsylvania, USA

Learned database systems address several weaknesses of traditional cost estimation techniques in query optimization: they learn a model of a database instance, e.g., as queries are executed. However, when the database instance has skew and correlation, it is nontrivial to create an effective training set that anticipates *workload shifts*, where query structure changes and/or different regions of the data contribute to query answers. Our predictive model may perform poorly with these *out-of-distribution* inputs. In this paper, we study how the notion of a *replay buffer* can be managed through online algorithms to build a concise yet representative model of the workload distribution — allowing for rapid adaptation and effective prediction of cardinalities and costs. We experimentally validate our methods over several data domains.

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

Additional Key Words and Phrases: learned database systems; workload shifts; replay buffer; online algorithms

ACM Reference Format:
Peizhi Wu and Zachary G. Ives. 2024. Modeling Shifting Workloads for Learned Database Systems. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 38 (February 2024), 27 pages. https://doi.org/10.1145/3639293

1 INTRODUCTION

Cost and cardinality estimation in DBMSs has advanced substantially over the decades, thanks to summary statistics (histograms, sketches, etc.) built over attributes. However, such approaches typically assume independence of attributes, and thus suffer in the presence of correlation. Unfortunately, efforts to capture correlations through multidimensional histograms have been shown to not simultaneously provide compactness, computational tractability, and accuracy [13]. Since heavily skewed and correlated data results in poor-quality estimates [43], heuristics have been developed to adapt from query workloads, e.g., through compensation factors [52]. Recently, more systematic frameworks, which follow optimal strategies under certain assumptions, have been proposed to incorporate machine learning models into query processing. Most research in this area (other than purely data-driven cardinality estimators [30, 74]) is workload-driven, i.e., leverages previous queries and outcomes (cardinality, cost) to train or improve the ML models. For example, learned cost estimators [51] build an ML model of the data instance by monitoring previous query executions to predict the costs of incoming queries, based on which learned query optimizers [49] enhance the plan generation strategies and reduce the query execution time. Indeed, this paper focuses on such workload-driven learned DB systems, targeting learned components such as cost and cardinality estimators.

Authors' addresses: Peizhi Wu, Computer and Information Science Department, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA, USA, pagewu@cis.upenn.edu; Zachary G. Ives, Computer and Information Science Department, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA, USA, zives@cis.upenn.edu.

2836-6573/2024/2-ART38 $15.00
https://doi.org/10.1145/3639293

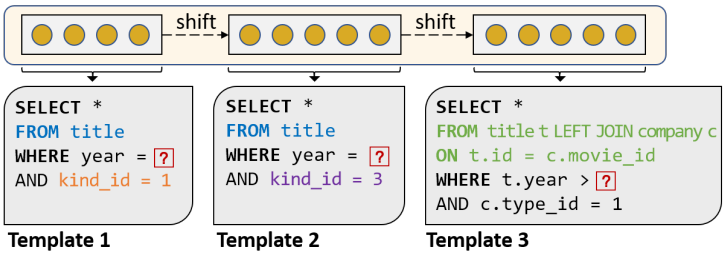


Fig. 1. An example of shifting workloads on IMDb

Most existing work implicitly focuses on scenarios where the query workload is *stationary*, *i.e.,* the structure of the query and the subset of the data queried are consistent between training and production workloads. In fact, the distribution of queries and data in workloads may shift over time [31], resulting in mis-estimates. This is sometimes termed "out-of-distribution" error.

Example 1.1 (Workload shift). Refer to Figure 1. Suppose the workload initially consists of 4 queries that can be considered to instantiate some SQL Template 1, which queries the Internet Movie Database for movies with `kind_id=1` (from distributors), and filters by year. Subsequently, 5 queries instantiate Template 2, which is similar but instead looks at movies from special effects houses (`kind_id=3`), which are sparser and have a different distribution of values. Finally, 5 queries are drawn from Template 3, which joins two relations. Most likely, a learned DB component trained on the former nine queries would produce poor results on the last five queries of Template 3, since the model only learned the information of the two classes of table `title`.

Of course, real workloads may not follow such a simple scenario: rather than discrete steps between different classes of queries, we may instead have a mix; the same query template might produce very different cardinalities with different values; there may not be a set of templates at all, but rather broad variation among queries. Nonetheless, the moment our system starts to encounter queries from outside its learned distribution — its model will suffer from error. The key question is *when and how best* to re-train the model.

The simplest idea would be retraining on the five queries of class 3. However, this would cause *catastrophic forgetting* [35], in which the learned DB component forgets the knowledge of the previous classes (*e.g.,* class 2). Now, if the learned DB component again encounters queries of class 2, it could make inaccurate estimates. Another naive idea is retraining with the latest k queries (k is larger than the number of new queries). This introduces challenges in setting the value of k, and again, this way the learned DB component might forget the knowledge of prior, older query classes.

One may naturally think of retraining with *all* previous queries. However, under heavy enterprise workloads, the workload history can be very long, resulting in a time-consuming retraining step — during which the learned DB component will continue to yield bad execution plans until it can adapt. More importantly, as we show later, this approach does *not* necessarily achieve a good model performance due to *workload imbalance*, a phenomenon in which the number of queries is nonuniform across query classes.

Therefore, the challenges of retraining a learned DB component over shifting workloads are two-fold. First, we aim to learn the best ML model for the DB component. Specifically, the learned DB component is required to learn new information from new queries while retaining the knowledge of previous queries. Second, it should rapidly react to new information in order to ensure the next query gets an accurate prediction or the best plan.

In this paper, we study the problem of *which query instances or summaries to remember* to build the best predictive model of complex, correlated data — even if the workload shifts to new portions of the overall data instance where we have poorer information or workload imbalance exists. We make the following contributions.

- We propose a general framework, based on a ShiftHandler, that handles workload shifts for learned database systems.
- We develop mechanisms for detecting workload shifts, that determine when a learned database system should retrain its model.
- We adapt ideas from *Bayesian non-parametric models* and the *facility location problem* to create the algorithm that provides optimal online clustering of workloads, within an approximation.
- Based on the online clustering algorithm, we develop techniques for managing the *replay buffer*, considering clusters of queries, balance across clusters, and model loss.
- We validate that maintaining a *replay buffer* of workload samples, managed using the above strategies, yields rapid retraining while also developing a model that is highly accurate over shifting workloads, even in the presence of workload imbalance.

2 PRELIMINARIES AND DEFINITIONS

We briefly review approaches to cost and cardinality estimation in query processing, including the notion of workload (query)-driven learned database systems. We then formalize the definition of *workload shifts* and *workload imbalance* in learned database systems, which are the problems we address in this paper.

2.1 Workload-Driven Learned Database Systems

Traditional query optimization can be effective when it is based on summary structures on table attributes that have been computed offline (e.g., histograms, and sketches). Unfortunately, correlated predicates (especially over skewed data) are difficult to predict with single-attribute synopses, often resulting in inaccurate estimates and poor plans [32]. This motivated research into multi-attribute structures such as join synopses [2] and multidimensional histograms [1]. Yet these structures imposed significant up-front costs, motivating later study of *opportunistic methods*, which bootstrapped off actual results from executed queries and views [12, 13]. In a similar spirit, workload results have also been used to make adjustments to correlated predicates [52].

In order to adapt in a more principled fashion, recent *learned database systems* replace traditional database components (*e.g.,* the cardinality estimator and underlying summary structures, query optimizer, and more) with ML models (*e.g.,* MSCN [37], Tree CNN [51], HAL [47], and more), in the hope of improving the performance of database components as the system learns to better estimate performance from past queries. Among existing proposals for learned database systems, most are workload-driven (or workload-aware), which means they are trained from and optimized for a specific workload. Indeed, this paper focuses on this setting, where we assume *a database instance with skew and correlation among attributes but variation in both the subsets of data queried and the structure of queries.* Ultimately, this means that models built over past queries (including models of skew and correlation) may not accurately represent more recent queries. We can formalize the notion of a workload-driven, learned database system and apply it to various components, such as cost or cardinality estimators, as follows.

Definition 2.1 (Workload-Driven Learned Database Systems). Let $\mathcal{CP}$ be a database component we want to optimize, which makes essential predictions (*e.g.,* predicting cardinalities) or decisions (*e.g.,* choosing execution plans) for queries. In the scenario of learned database systems, a record (log) of previously executed workload $\mathcal{W}$, which consists of a set of n queries $\{q_i\}_{i=1}^{n}$ and their

outcome $\{o_i\}_{i=1}^{n}$ (including cardinality, cost, and more, which we refer to as *label*) is available. Workload-driven learned database systems aim to train a machine learning model $\mathcal{M}$ to make predictions/decisions for $\mathcal{CP}$, by leveraging the historical workload $\mathcal{W}$.

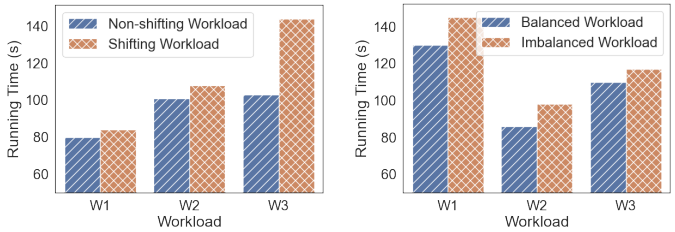
2.2 Workload Shifts

In the machine learning literature, if the distributions of the data change over time, it is well-known that we should adapt the model accordingly [24, 38]. Recall that in the toy Example 1.1, a learned DB component trained on queries conforming to Templates 1 and 2 would likely make bad predictions for queries conforming to Template 3, since it had learned nothing about the join of tables `title` and `company`. Yet, many recent systems assume that *training and production workloads are drawn from the same distribution*, which we term a *stationary workload assumption*. Unfortunately, although not in as discrete a fashion as Example 1.1, real workloads are seldom static, but change over time, *i.e.,* the workloads are mixed and evolving. Given that our training is over a small sample of the space of queries and predicates, we have an imperfect model of how future queries might perform over unseen parts of the underlying database instance. For instance, users of a database may query using a join graph that the learned DB component has not been trained for (even though the database instance itself might have remained static). Even if the join graph has been explored by the learned DB component, users may also include predicates or value ranges for which it has incomplete information. Moreover, in cases where queries in the new workload significantly deviate from the workload to which the model was previously adapted, it is crucial to re-adapt the model to the new workload. Despite the model having encountered a similar querying pattern in the past, re-adaptation is necessary to maximize its performance on the specific characteristics of the new workload. Thus the learned DB component needs to strategically adapt to the new workload. Otherwise, one can anticipate the degradation of model performance.

Definition 2.2 (Workload Shifts in Learned Database Systems). Given a workload-driven machine learning model $\mathcal{M}$ for a database component $\mathcal{CP}$ of a database system $\mathcal{DB}$, workload shifts refer to the phenomenon when the distribution of the querying pattern of workload $\mathcal{W}$ on $\mathcal{DB}$ is changing over time. Specifically, the query join graph and/or the distribution of query predicates may shift with time, which causes the portions of database instances that contrite to query answers to change over time. The definition arises naturally in real-world applications.

Note that retraining from scratch for the learned database component $\mathcal{CP}$ can result in a long retraining interval, especially across enterprise workloads — and thus a significant number of queries may start before the model is updated. If retraining only using new queries, the learned database component $\mathcal{CP}$ would suffer from substantial degradation in prediction accuracy and decision quality.

Preliminary Evaluation. In order to verify that our expectations are correct, we consider an existing benchmark of query optimizer predictive accuracy [43], which synthesizes queries from various "JOB" classes. We perform cost estimation using the learned query optimizer Bao [49] integrated into PostgreSQL. We generate two sets of 50 queries from the same JOB class T and concatenate them together, as a non-shifting workload. Then we insert a set of 50 queries from another class T' in between the two query sets of T to form a shifting workload for comparison. For the two types of workload, we retrain Bao at the end of each set using the 50 queries from that set and report the running time of the last query set of T. The results (averaged across three repetitions) are shown in Figure 2a. Bao's running time on shifting workloads is longer than the relative non-shifting workloads by 17% on average. Hence, we conclude that workload shifts significantly weaken the performance of learned database systems on old workloads.



(a) Effect of shifting workloads (b) Effect of imbalanced workloads

Fig. 2. Preliminary evaluations of learned query optimizer on shifting and imbalanced workloads

2.3 Workload Imbalance

To get effective machine learning models, we should use training data that is carefully selected to be balanced with regard to class labels, or else the model may be biased towards certain predictions. Moreover, various hyperparameters must be set to balance the *bias-variance tradeoff* to fit but not *overfit* the training data.

Most existing learned DB components in fact assume that *training workloads are (near) perfectly balanced across query classes.* However, training on an imbalanced set of instances may result in poor performance on certain classes of inputs. In pure machine learning classification problems, a data scientist seeks balance in the instances belonging to different classes. Again, back to Figure 1, if we increase the number of queries of class 2 to 1K and train an ML model on queries of the former two classes, one would naturally suspect the ML model would degrade for class 1 since the much-more-prevalent class 2 dominates the training set.

Definition 2.3 (Workload Imbalance in Learned Database Systems). Given a workload-driven learned database component, workload imbalance refers to the problem that the distribution over the query features (*e.g.,* query class, predicates) is skewed. In other words, queries of one specific range of query features are significantly fewer than those of the other range of query features.

Preliminary Evaluation. We first randomly select 4 classes from the JOB classes and use them to generate two sets of 80 queries (20 queries for each class). We concatenate them to compose a balanced workload. Next, to form the comparative imbalanced workload, we replace the first query set with another set of 80 queries in which the distribution of the number of queries of each class is skewed. Similar to § 2.2, we retrain Bao for every 80 queries with the same number of experiences. The running time of the second set of 80 queries is evaluated. The experimental results on three such workloads are shown in Figure 2b. We observe an average 11% increase in Bao's running time from balanced workloads to imbalanced counterparts. This indicates that learned database systems could significantly degrade for minority queries whose query classes are under-represented in imbalanced workloads.

2.4 Workload- *vs.* Data-driven Learned Databases

We note that *data-driven* learned models exist for cardinality estimation and are unaffected by workload shifts — so it is important to clarify why we focus on workload-driven models. Data-driven methods have notable limitations: to our knowledge, existing data-driven methods are limited to approximate query processing and predicting cardinalities. Moreover, existing data-driven learned cardinality estimation models are limited to SPJ queries with limited join types (PK-FK, FK-FK) [36]. Yet, many modern learned DB tasks model factors beyond cardinality, such as running time (e.g.,

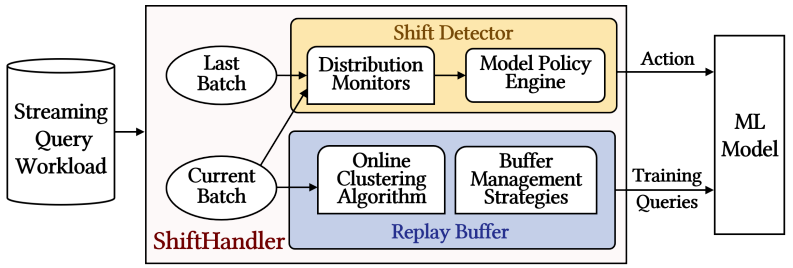


Fig. 3. Overview of ShiftHandler

[2]), CPU usage [3]. Such tasks inherently require workload- or query-driven methods, because they are modeling how the data correlates to query-based parameters such as costs. Moreover, as discussed in [55], even for cardinality estimation, query-driven methods have low model complexity and faster inference and are easily extensible to more complex queries. Query-driven models have significant value and cannot be fully replaced by data-driven models.

2.5 Desiderata and Problem Formulation

Evidence suggests learned DB systems should balance three desiderata:

(1) *Long-term memory*: The system, even when retrained, should retain the core information obtained from prior queries for producing good-quality query execution plans.
(2) *Plasticity*: The system should adapt quickly to shifts in the workload and use new information to improve its predictions.
(3) *Tolerance to imbalance*: When different classes of queries occur with different frequencies, the system should learn accurate models of all of these *imbalanced* classes.

This paper develops strategies for managing a *replay buffer* (which stores query instances for future retraining) for achieving the three desiderata. Given a workload-driven machine learning model $\mathcal{M}$ for a database component $\mathcal{CP}$ of a database system $\mathcal{DB}$, we maintain a small replay buffer that efficiently and effectively models up-to-date workloads to $\mathcal{M}$ for retraining, such that $\mathcal{M}$ maintains *long-term memory* and is of *high plasticity* over shifting workloads, even with the presence of *workload imbalance*.

3 MANAGING RETRAINING VIA A ShiftHandler

Overview. To address the problems of managing a learned DB component (e.g., cardinality or cost estimator) to shifting workloads, we propose a module that (1) manages the specific set of training data that will be used for learning, and (2) determines when to trigger relearning. We refer to this as the ShiftHandler: As illustrated in Figure 3, the inputs to ShiftHandler are a streaming query workload and the target ML model of a DB component. The module works as a middleware between the query workload and the ML model. It digests the streaming queries from the workload and makes the action decision. If training is triggered, it provides the training queries to the ML model. The ShiftHandler has two components:

(1) The *shift detector* (§ 4) works on the current query batch and the last query batch. The current batch of queries is the set of queries that arrived after the last training action, and the last query batch contains queries that arrived before the last training action and after the second last training action. For every l_d query (l_d can be set by DBA), it detects potential workload shifts between the two batches using the **distribution monitor**. The **model policy engine** receives

the statistics from the distribution monitor and triggers the proper actions (periodic training or training for workload shifts) for the ML model.

(2) The *replay buffer manager* with a query limit m operates on the current batch of queries, which is shared with the shift detector. It takes the streaming workload as input and maintains a set of representative and useful queries. Indeed, the maintenance of this replay buffer, in terms of choosing representative instances and incrementally maintaining them in a theoretically optimal way, is our core contribution.

In accordance with the three desiderata, the query limit m ensures *plasticity* and the **online clustering algorithm** (§ 5) provides *long-term memory* since it effectively models the workload characteristics. Moreover, the **buffer management strategies** (§ 6) offer *tolerance to imbalance* by considering query cluster diversity and further improve *long-term memory* using model loss.

When the model policy engine triggers a training action, the ShiftHandler immediately delivers a training set composed of current queries *as well as* historical queries from the replay buffer, to the target ML model's retraining phase. Retraining is achieved by optimizing the combined *loss* (a cumulative measure of prediction error) from the mix of current and historical queries. Suppose the model losses on the current batch of new queries and on the historical (replay) queries are $\mathcal{L}_c$ and $\mathcal{L}_s$, respectively. The model is trained to minimize a convex combination of the two losses

$$\mathcal{L} = \psi \mathcal{L}_c + (1 - \psi)\mathcal{L}_s, \tag{1}$$

where $\psi \in [0, 1]$ is a parameter controlled by the model policy engine. ψ trades-offs the relative importance between the current batch and replay buffer (higher ψ focuses more on the current batch). After completing the training query delivery, ShiftHandler digests the current batch, flushes them to the last batch, and empties the current batch for receiving newly arrived queries. Note that the ShiftHandler framework is general and can be employed in various learned workload-driven DB models defined in Definition 2.1.

Handling Temporary Workloads. ShiftHandler suggests a gradual change in the workload characteristics so that previous workloads can help the model make better predictions for future workloads. However, when temporary workloads (which are completely different from previous queries, such that no knowledge from prior workloads can be used) occur, it would be more efficient to train a new model that could temporarily replace the existing one. Here, the extra instances in the replay buffer will not substantially help for retraining, but the ShiftHandler can still efficiently detect temporary workloads and trigger actions at the right point. This is because detecting temporary workloads needs to compare to previous queries; such queries are summarized in the replay buffer.

Handling Data Shifts. ShiftHandler assumes the underlying data to be static. However, one could augment ShiftHandler with a data shift monitor (e.g., using techniques from DDUp [41]) and update query labels in the replay buffer when a data shift is detected. Specifically, DDUp is a framework that provides updatability for *data-driven* models facing *data* shifts. It consists of two components — 1) a statistical-testing-based data shift detector and 2) an approach to efficiently update data-driven models. The model-updating approach cannot be applied in ShiftHandler since we focus on query-driven models, but ShiftHandler can use its shift detector to accurately and efficiently detect data shifts.

When a data shift is detected, the labels for queries in the replay buffer may become obsolete. Consequently, we need to re-obtain their labels by re-running them, possibly through a DBMS query. Notably, the Warper [44] system also adopts this approach to handle data shifts for query-driven models. Here, a side benefit of ShiftHandler is that since the replay buffer is designed to store useful queries within a *limited* capacity, we could minimize/bound the time required for query re-execution. Another possible technique that ShiftHandler could leverage to further reduce the

number of queries needed for re-execution is using data provenance [14] to filter out queries whose results are *not* affected by data changes. To limit the scope of our paper, we leave these as future work.

4 DETECTION OF WORKLOAD SHIFTS

The shift detector in ShiftHandler detects possible workload shifts by comparing the current query batch and the last query batch. If a significant dissimilarity arises between the last batch, to which the ML model has been adapted, and the current batch in terms of workload characteristics encompassing the distributions of query features and labels, a workload shift is detected and the ML model needs retraining to learn from/adapt to the new query batch. The batch size is bounded by the periodic training interval (see § 4.2), thus the detection is efficient.

4.1 Distribution Monitor

Our distribution monitor operates on query features rather than labels for their empirical performances (will be studied in § 7.3). The goal of distribution monitoring is essentially comparing the query feature distributions of the current query batch (denoted by P) and the last query batch (denoted by Q). In fact, this is a fundamental problem in statistics. One principled method would be *two-sampled tests* [42], which summarizes the differences between two samples ($q_1 \sim P, q_2 \sim Q$) from each distribution into a test statistic. The real-valued test statistic is then used to accept or reject the null hypothesis H_0, *i.e.,* $P = Q$.

Two-Sample Testing. The idea is to verify if two samples, $q_1 \sim P, q_2 \sim Q$, are drawn from the same distribution. We define that a smaller statistic indicates that H_0 is more likely to be true. The two-sample tests accept H_0 if the statistic is sufficiently small or reject H_0 if the statistic is sufficiently large. A significance level $\alpha \in [0, 1]$ is provided as input, and the test computes the two-sample test statistic $\hat{t}$. Based on the result, it computes the p-value by $\hat{p} = P(T \geq \hat{t}|H_0)$, representing the probability of the two-sample test obtaining a statistics T larger than or equal to $\hat{t}$, given the hypothesis H_0. Finally, it accepts H_0 if $p \geq \alpha$, and rejects H_0 otherwise. We can adapt the univariate two-sample tests to the case when queries are one-dimensional. We use the well-known Kolmogorov-Smirnov (KS) test [39, 63] for its efficiency and effectiveness. KS test is a non-parametric test that summarizes the difference in the empirical cumulative distributions of two samples in one dimension.

Multiple KS Tests with Bonferroni Correction. However, typically query representation (or encoding) is high-dimensional in learned DB components. It would be computationally inefficient to consider all possible feature correlations in the KS test. Inspired by [59], we apply a simple yet practical approach to solve this. We first employ the KS test for each of the features, which assumes the features are independent. Then, to account for feature correlations while ensuring efficiency, we leverage a conservative aggregation approach, the Bonferroni correction, to remove the strong assumption to some extent for its efficiency and empirical performance [59]. Bonferroni correction works by rejecting the null hypothesis if the minimal p-value for all features is less than α/K (for K features).

4.2 Model Policy Engine

The model policy engine controls the actions taken by the ML model based on the statistics it receives from the shift detector. Possible actions include 1) periodic training and 2) training for workload shifts. Action 1 (periodic training) triggers for every l_p query, where l_p is a hyper-parameter to the system. Note that l_p is typically larger than the detection interval l_d of the shift detector. Besides, if the statistics indicate that a workload shift is happening, the model policy engine *immediately*

triggers action 2 (training for workload shifts) for quick adaptation. When no action is triggered, the ML model keeps its model weights and functions as normal.

5 ONLINE CLUSTERING OF WORKLOADS

We aim to design a replay buffer that generates a compact summary for shifting or even imbalanced workloads in this section. The replay buffer is the core of ShiftHandler as it produces the replay instances that directly affect the training and performance of the target ML model. As with the ML model updates, the buffer should also handle the streaming nature of query workloads. A naive approach would be reconstructing the buffer using all previous queries each time. However, again this approach would be neither space-efficient; nor time efficient. Specifically, the replay buffer is supposed to provide the replay queries to the ML model *on time* whenever retraining is triggered. This could be impossible if the workload is heavy (in which reconstructing the buffer takes too long) and the workload shifts frequently (in which the buffer is required to process all previous queries in a small window). As a result, the replay buffer needs to be constructed in an online fashion.

A naive method for generating the replay buffer in an online fashion would be reservoir sampling (RS) [69], which uniformly samples m queries from a streaming workload by replacing a random query in the replay buffer with a new query with probability m/n, where m is the buffer size, and n is the total number of queries seen so far. Although it is efficient and works well for perfectly balanced data, reservoir sampling suffers under imbalanced workloads because it may not capture the diversity of a workload, and the minority could be under-represented by the RS scheme.

To capture the feature structure of a workload, clustering would be a natural idea to select the most representative queries in high-dimensional query space. Therefore, we aim to provide an efficient, effective, and provable clustering-based strategy for streaming workload sampling in query space. Before running clustering algorithms, we need to represent/encode the queries so that the distances (or similarities) between queries can be measured. We justify our choice of query representation in § 5.1. We introduce two fundamental challenges in managing the replay buffer and show our solution for addressing the first challenge in § 5.2. Then to deal with the second challenge, we first show the connection between online DP-Medoids and online facility location (a well-studied problem in theoretical computer science) by a reduction. Based on the reduction, we present a provably optimal algorithm for online DP-Medoids that enjoys theoretical guarantees in § 5.3.1. We initially consider an unbounded query buffer in this section.

5.1 Query (Plan) Representation

ShiftHandler aims to provide a general framework for modeling shifting workloads. Given that the query features relevant to various database tasks are different (*e.g.,* cardinality is based on the logical query expression whereas cost must consider the physical query plan), it is unrealistic to propose a unified query representation that can fit all purposes. We thus directly leverage the query representation method suggested by the target ML model.

Levels of query encodings. There are three types of query encodings that we could extract from the target ML model, *i.e.,* feature-level, embedding-level, and gradient-level query encodings.

- **Feature-level query encoding** is the vector representation of queries *before* any learnable embedding layer. Note that we should fix the encoding dimension across queries to make them comparable with any distance metric. For example, for MSCN [37] in the cardinality estimation task, the feature-level encoding could be the concatenation of the outs of three set modules.
- **Embedding-level query encoding** is the intermediate query vector representation *after* some learnable embedding layers (normally the first learnable embedding layer). For example, the

embedding-level query encoding of Tree-CNN [49] could be the output of the dynamical pooling layer.
- **Gradient-level query encoding** is the vector that concatenates the gradients of all the model weights. The idea behind gradient-level query encoding is that it could indicate the contribution of a query to the model training when applying gradient-based optimization methods such as stochastic gradient descent (SGD).

The default choice. We opt for feature-level query encoding over the other two, for three reasons. First, feature-level query encoding is model-agnostic and is compatible with any ML model, which fits the goal of a general framework. Second, we experimented with the two alternatives, using MSCN to perform cardinality estimation; and the performance was notably worse than with feature-level query encoding. We believe that it is likely because the learned embeddings (or gradients) of a query might be divergent (or inconsistent) across various training phases in a streaming workload. Thus, two queries that are close in embedding (or gradient) space in previous training phases could be distant in the same space in the later training phase. Third, computing gradients is much more computationally expensive than computing the other two (features and embeddings). (We empirically observed that computing gradients is more than 100× slower than computing query features.)

5.2 K-Medoids with Adaptive Cluster Number

Our strategy for managing the replay buffer then must address two key challenges:

- **No prior knowledge of the number of clusters.** Since 1) there is no prior knowledge of the number of clusters and 2) manually setting the cluster number could result in sub-optimal clustering performance, we need to adapt the cluster number from the query workloads by respecting the workload characteristics.
- **Accurate online clustering for streaming query workloads.** One would naturally question the quality of the clusters from online clustering methods that can only make one pass over the workload: this could lead to sub-optimal results compared to a multi-pass, offline algorithm.

With the proper vectorized query representation, the similarities (or distances) between queries can be measured so that we can run clustering algorithms to find out the most representative and diverse queries in a workload.

K-Medoids. K-Means [48] and K-Medoids [58] are among the two most studied clustering problems in the literature since they are simple and intuitive for query encodings in Euclidean space. Both problems produce a set of k (which is given as input) queries as cluster centers, and each query is assigned to the cluster center nearest it in Euclidean space. Additionally, both algorithms attempt to minimize the sum of distances (or dissimilarities) between queries and the cluster "center" in which queries are labeled. The major difference between the two problems lies in selecting "centers". In K-Means, a cluster "center" is the cluster centroid which is the mean of all queries in the cluster and is not necessarily an actual input query. In contrast, the cluster "center" in K-Medoids refers to the medoid, which is a query that *belongs to the dataset*, calculated by minimizing the sum of pairwise distances between the selected center and the queries in the cluster. Note that both problems were proved to be NP-hard [16, 34]. Therefore, most efficient algorithms for them are of approximation algorithms whose objective function value is within a factor of the value of the optimal function.

We choose K-Medoids for the replay buffer instead of K-Means for at least three reasons. First, we do not have actual labels for queries not in the workload since they were not executed. K-Medoids can bypass this issue since all medoids are selected from the actual queries in the workload. This can offer interpretable cluster representatives. Second, K-Means is specifically designed for Euclidean distances, whereas K-Medoids supports arbitrary distance measures. This allows users to choose

the best distance measure for their problem at hand. Third, K-Medoids is more robust to outliers because the computation of centers in K-Means can be easily dominated by outliers [66] (will be evaluated in § 7.5), which could appear in imbalanced workloads. Therefore, K-Medoids produces more robust estimates of representative points than K-Means.

The objective function of K-Medoids is

$$\arg\min_{C,|C|=k,C\subseteq\mathcal{W}} \sum_{q\in\mathcal{W}} \min_{\mu\in C} D(q,\mu) \tag{2}$$

where q is a query in the workload $\mathcal{W}$, C is the set of centers[1] learned from K-Medoids and $D(q,\mu)$ is an arbitrarily chosen distance between q and μ, *e.g.,* squared Euclidean distance which we choose, *i.e.,* $D(q,\mu) = \|q-\mu\|_{l_2}^2$, but of course, K-Medoids is compatible with other distance functions as well. Similar to K-Means, common practice for optimizing the objective function is an Expectation-Maximization (EM)-style algorithm, in which we iteratively update the centers and the cluster assignments for all queries. The iterative process continues until coverage. Note that the EM algorithm typically needs to make multiple passes of the workload in order to reach a local optimum [71].

Nevertheless, the number of clusters is not known *a priori* for a streaming workload, so off-the-shelf K-Medoids algorithms cannot be directly applied here without any modifications.

Bayesian Non-Parametric Priors and DP-Means. Bayesian non-parametric (BNP) models [57] can be used to adapt the number of clusters. BNP models, such as Dirichlet process mixture [8], are infinite mixture models which do not fix the number of mixture components in advance. Instead, the number of components is inferred from workload. This way BNP models adapt the number of components to avoid overfitting (too many components) or underfitting (too few components).

DP-Means [40] is the pioneering work that combines the Bayesian non-parametric priors with K-Means, achieving the adaptivity of cluster number for K-Means. However, two major approaches (Gibbs sampling and variational inference) for inferring BNP models are much more computationally expensive than the simple EM algorithm used for the original K-Means. To ease this issue, Kulis and Jordan [40] propose a deterministic and scalable hard clustering algorithm through Bayesian nonparametric for the Dirichlet process (a BNP model). In fact, the hard clustering algorithm is similar to the original K-Means with EM, except that if the distance between a query and its nearest cluster center is larger than a threshold λ, a new cluster should be created on the query. Kulis and Jordan [40] also prove that the hard clustering algorithm optimizes the original objective function of K-Means (similar to E.q. 2, but no longer requiring $C \subseteq \mathcal{W}$), plus the penalty for the number of clusters created. For details regarding BNP models, please refer to [40].

DP-Medoids. Inspired by the way DP-Means incorporates BNP priors (Dirichlet process) into K-Means, we propose the DP-Medoids algorithm, which empowers K-Medoids with the ability to adapt the cluster number from the workload. The offline version of DP-Medoids is shown in Algorithm. 1. DP-Medoids iterates over all queries in a workload and computes the minimum distance to all existing cluster centers for each query. The main difference from DP-Means is, at the end of each iteration, DP-Medoids updates the center of each cluster by FindMedoids(·), which takes a set (cluster) of queries as input and outputs the query that minimizes the sum of pairwise distances between all queries in the cluster to the query as the new center. It repeats the procedure until convergence.

Furthermore, by following the Proof for Theorem 3.1 in [40], one can easily show that the underlying objective (cost) function for DP-Medoids is E.q. 2 plus the number of clusters as penalty, *i.e.,*

[1] "center" and "medoid" are interchangeable thereafter

$$\arg\min_{C, C \subseteq \mathcal{W}} \quad \lambda k^* + \sum_{q \in \mathcal{W}} \min_{\mu \in C} D(q, \mu), \tag{3}$$

where $k^* = |C|$ is the cluster number learned from DP-Medoids.

Algorithm 1 Offline DP-Medoids

Input: λ, a set of n queries $\{q_i\}_{i=1}^n$
Output: Medoids $\{m_i\}_{i=1}^{K^*}$, where K^* is adapted from the workload

```
K = 1, m_1 = FindMedoids({q_i}_{i=1}^n), (z_i)_{i=1}^n = 1 //Initialization
while Not Coverged do
    for i = 1 to n do
        z_i = arg min_{k∈{1,...,K}} ||q_i − μ_k||_{l_2}^2
        if ||q_i − μ_k||_{l_2}^2 > λ then
            K = K + 1, μ_K = q_i, z_i = K
    for k = 1 to K do
        m_k = FindMedoids({q_i | z_i = k})
```

Time Complexity. In each iteration, computing the distance between each query and existing centers takes $O(nK^*)$ time, while updating the cluster centers by FindMedoids$(\cdot)$ takes $O(n^2K^*)$ time since every FindMedoids$(\cdot)$ should compute the distances for all pairs in the input cluster. To sum up, the total time complexity is $O(n^2TK^*)$ where T is the number of iterations.

5.3 Provable Online DP-Medoids

As discussed before, the major obstacle for applying the offline version of DP-Medoids in ShiftHandler lies in that retraining DP-Medoids is computationally expensive for a heavy workload of size n since the time complexity is quadratic in n. One would resort to an online version of DP-Medoids which only makes a single pass of the workload and does not require retraining as new queries come in. However, the design of online DP-Medoids must be careful since it should not lose too much performance compared to the optimal one, *i.e.,* the objective function value for online DP-Medoids should be provably within a factor of the value of the optimal one.

Unfortunately, to our knowledge, there is no existing work that studies such an online version of DP-Medoids with theoretical guarantees. Several relevant papers are [19, 26, 56], in which [19, 56] are regarding DP-Means and [26] focuses on K-Median without Bayesian nonparametric priors. Moreover, while the two proposals on DP-Means achieve online learning of nonparametric clustering, none of them provides theoretical guarantees for their algorithms. Inspired by a few theoretical works [11, 61] on K-Means, we introduce a randomized solution to online DP-Medoids by an approximation-preserving reduction to online facility location. Although the solution may appear strikingly simple, they work surprisingly well. More importantly, we can analyze their approximation ratios with respect to the optimal one via the reduction, which justifies our solutions in a principled way. In addition, due to their simplicity, they are easy to state and implement in real systems.

(Online) Facility Location. We formally define the problem of facility location and then extend it to the online version.

Definition 5.1 (Facility Location). Given a metric space along with a multiset of demand points $X = \{x_i\}_{i=1}^n$, a facility cost f_i for each point x_i (the cost for opening a new facility on x_i), and n valid distance function Dist$(\cdot, \cdot)$. The goal is to seek a set of facility locations $F \subseteq X$ from the demand

points so as to minimize the sum of facility cost and assignment cost (the distance of a demand's location to the nearest facility). Thus, the total cost for facility location is

$$J = \underbrace{\sum_{z \in F} f_z}_{\text{facility cost}} + \underbrace{\sum_{i=1}^{n} \min_{z \in F} \text{Dist}(x_i, z)}_{\text{assignment cost}} . \tag{4}$$

Facility location is an intensely studied problem in both the operations research and the computer science literature, which is motivated by many real applications, *e.g.,* constructing a sensor network or computer network.

The *online* version of the facility location (OFL) problem is defined in [53]. In OFL, the demand points are not known in advance and arrive one at a time. They are irrevocably assigned either to an existing facility or a new facility on it. OFL is also motivated by real-world considerations. For example, telecommunication networks are upgraded (by purchasing new cables or new servers) incrementally to accommodate new clients (demands) since reconfiguring the entire network whenever a new client arises is highly infeasible. Note that the total cost for OFL is the same as Equation. 4.

Approximation-Preserving Reduction. We then show that the problem of online DP-Medoids is reducible to the problem of online facility location via an approximation-preserving reduction.

Theorem 5.2. *Online DP-Medoids is reducible to OFL via an S-reduction [15] (the strictest type of approximation-preserving reduction, where the two instances' optima have the same cost).*

Proof. Let us consider an instance $\varphi_{\text{DP-Medoids}}$ of online DP-Medoids with n queries $\{q_i\}_{i=1}^n$, hyperparameter λ and a distance measure $D(\cdot, \cdot)$. We construct the corresponding instance φ_{OFL} of OFL in the following way: 1) let the demand points X be $\{q_i\}_{i=1}^n$. 2) $f_i = \lambda$ for all demand points x_i. 3) the distance function $\text{Dist}(\cdot, \cdot)$ is chosen as $D(\cdot, \cdot)$. In fact, the resulting OFL instance φ_{OFL} is a special type of OFL with uniform facility costs [53]. Interestingly, from E.q. 4 and E.q. 3, acute readers may also observe that the cost function for the resulting $\varphi_{\text{DP-Medoids}}$ becomes identical to the objective function for φ_{OFL}.

Now, for each facility configuration F of φ_{OFL} and the cost $J(\varphi_{\text{OFL}}, F)$ for the configuration F, we consider an identical mapping from F to C (centers in online DP-Medoids), *i.e.,* $C \equiv F$. Then suppose F^* is the optimal solution (facility configuration) for φ_{OFL}, it is easy to see that the corresponding C is also the optimal solution for $\varphi_{\text{DP-Medoids}}$ with the same cost of $J(\varphi_{\text{OFL}}, F^*)$ since the objective functions for $\varphi_{\text{DP-Medoids}}$ and φ_{OFL} are essentially equivalent. □

The approximation-preserving reduction suggests that we can convert an instance $\varphi_{\text{DP-Medoids}}$ of online DP-Medoids to an instance φ_{OFL} of OFL, apply the solver for φ_{OFL} and recover the solution for $\varphi_{\text{DP-Medoids}}$ simply by an identical mapping. We are surprised that to the best of our knowledge, we are the first to establish the straightforward but very useful connection between DP-Medoids and OFL, as it allows us to design provable algorithms for DP-Medoids by leveraging theoretical work on OFL as follows.

5.3.1 ***Algorithm for Online DP-Medoids.*** We propose our randomized algorithm for online DP-Medoids (called ODMedoids) from [53] in Algorithm 2. The algorithm maintains current cluster centers C throughout the online learning process. For each new query q, it first computes the minimal distance d between q and all centers in C (line 1). Then with probability d/λ (λ is cluster penalty parameter), we create a new cluster on q (line 3-4). Otherwise, q is labeled with the nearest cluster, and C is unchanged (line 5-6). Note that one consequence of the randomized solution is that even if $d < \lambda$, the new query q is still possible to create a new cluster.

Algorithm 2 Online DP-Medoids (ODMedoids)

Input: λ, new query q_i, and current cluster centers C
1: $d = \min_{\mu \in C} \|q_i - \mu\|_{l_2}^2$
2: Uniformly sample a number $i = \text{random}(0, 1)$
3: **if** $i < d/\lambda$ **then**
4: $\quad C \leftarrow C \cup \{q_i\}$ # Update C by adding q_i
5: **else**
6: $\quad$ Assign q_i to the nearest cluster # C is unchanged

Time Complexity. For each new query q, computing the distances between q and all centers in C takes $O(K^*)$ time, where K^* is the ultimate cluster number. Since ODMedoids does not require retraining from the beginning, the total running time for a new batch is $O(n^{new}K^*)$, where n^{new} is the number of queries in the new batch. which is way more efficient than offline DP-Medoids.
Approximation Ratio. Surprisingly, not only ODMedoids achieves impressive empirical performances (as presented in § 7), but also we can prove its approximation ratios theoretically through the reduction to online facility location. Typically, the theoretical performance of online algorithms is judged by competitive analysis [10] (competitive ratio), whose definition is basically identical to that of approximation ratio. An online algorithm is $c-$competitive if its cost is at most c times the optimal cost for the corresponding offline algorithm. Citing from [25, 53], ODMedoids can achieve a constant competitive ratio $O(1)$ with a factor of 8 when queries arrive in random order and not chosen by an adversary. This implies ODMedoids performs very well for most of the query orderings, thereby being practical for real workloads. Even for the case where an oblivious adversary creates the ordering, the competitive ratio for ODMedoids is $O(\log n / \log \log n)$.

6 BUFFER MANAGEMENT

Our near-optimal online workload sampling algorithm, ODMedoids, has not considered capacity constraints on the buffer. To achieve *plasticity*, we need to put a query limit on the replay buffer: if the buffer size is too large, I/O costs from permanent storage and training costs may be significant, resulting in long latencies to retrain. In this section, we propose two strategies for buffer management.

6.1 Goals

Our design of the first buffer management strategy, Cluster-Balancing Buffer Population (CBP), is derived from a few goals.

1. **Center Priority**. Each cluster from ODMedoids represents a querying pattern (or plan pattern if the encoding is done on the plan level), in which the center is the most representative query of the cluster. As such, centers would be prioritized to be stored over non-centers, but non-centers could also be stored as complementary instances if the replay buffer is not filled.
2. **Diversified Clusters**. When dealing with workload imbalance (e.g., queries from a class is significantly fewer than queries from another class), we have to keep the number of clusters stored in the buffer as much as possible as the clusters are a compact and diverse summary of the workload.
3. **Balanced Cluster Sizes**. We make the cluster sizes as balanced as possible to further mitigate potential workload imbalance.

6.2 Operations

Algorithm 3 RS_ASSIGN_CENTER(q_i, o_i)

```
1: if there are non-center instances in the buffer then
2:    Remove a non-center instance from the largest cluster at random, then perform
      DIRECT_STORE(q_i, o_i, True, −1)
3: else
4:    m_c ← # of existing centers
5:    n_c ← # of centers ever created
6:    Sample a number i = random(0, 1)
7:    if i < m_c/n_c then
8:       Remove a center instance at random
9:       DIRECT_STORE(q_i, o_i, True, −1)
10:   else
11:      Ignore (q_i, o_i)
```

Algorithm 4 Cluster-Balancing Buffer Population (CBP)

Input: A streaming workload $\{(q_i, o_i)\}_{i=1}^{n}$
Output: A replay buffer of size m

```
1: for i = 1 to n do
2:    I_c , c_i = ODMedoids(q_i, o_i)
3:    if the replay buffer is not filled then
4:       DIRECT_STORE(q_i, o_i, I_c, c_i)
5:    else
6:       if (q_i, o_i) is a new center then
7:          RS_ASSIGN_CENTER(q_i, o_i)
8:       else
9:          RS_ASSIGN_NONCENTER(q_i, o_i, c_i)
```

Given ODMedoids(q_i, o_i) we are able to find out if (q_i, o_i) is a new cluster or the existing cluster it belongs to. Specifically, for each (q_i, o_i), it will return two values, namely, I_c, and c_i. I_c indicates if (q_i, o_i) will be a new center and c_i represents the cluster ID of (q_i, o_i) if (q_i, o_i) is assigned to an existing cluster. It is easy to see c_i is valid only if I_c is False; otherwise, c_i can be any value (*e.g.,* -1).

We then implement three abstract operations (ops) related to storing a new query in the replay buffer. Ops 2 and 3 differ from op 1 in that they consider the case where the reply buffer is filled and an old query may be replaced by a new query, while op 1 does not.

1. **DIRECT_STORE(**q_i**,** o_i**,** I_c**,** c_i**)** directly stores (q_i, o_i) using the return values (I_c, c_i) of ODMedoids.
2. **RS_ASSIGN_CENTER**(q_i, o_i) (Algorithm 3) handles the case when the replay buffer is filled, requiring a decision whether to store a new center (q_i, o_i) or ignores it. We use the idea of reservoir sampling to keep independent and identically distributed (iid) centers for the streaming workload. *I.e.,* the chance for a center to be stored in the replay buffer remains consistent with other centers throughout the entire stream. With probability m_c/n_c, we replace a center instance chosen at random with instance (q_i, o_i) (line 6-9). Otherwise, (q_i, o_i) is ignored (line 11).
3. **RS_ASSIGN_NONCENTER**(q_i, o_i, c_i) assigns q_i as a non-center instance to cluster c_i or ignores it via reservoir sampling, which is similar to the implementation of RS_ASSIGN_CENTER. To enforce that sizes are balanced across clusters if c_i is not one of the largest clusters, we replace a random non-center instance from the largest cluster with (q_i, o_i). Otherwise, for the case where

there are non-center instances, again we use the idea of reservoir sampling to replace a random non-center instance in cluster c_i with probability m_n/n_n, where m_n and n_n are the numbers of non-center instances currently in c_i and ever created in c_i, respectively. Lastly, if there is no non-center instance, considering the priority of center instances we ignore (q_i, o_i).

Given the above operations, we sketch the full CBP algorithm in Algorithm 4. When the replay buffer is not used up yet, CBP simply populates the replay buffer with new instances (lines 3-4). After this stage, for each new instance (q_i, o_i), CBP first checks if the replay buffer query limit is reached (line 6). Otherwise, CBP calls RS_ASSIGN_CENTER(q_i, o_i) (line 7) or RS_ASSIGN_NONCENTER $(q_i, o_i.c_i)$ (line 9), depending on if (q_i, o_i) creates a new cluster.

6.3 Loss-Weighted Buffer Population

One may question the third goal of CBP in § 6.1, as clusters might exhibit different sample complexities [28]. To be specific, the number of query samples necessary for effective model training may naturally vary across clusters. For example, in the context of cardinality estimation where the underlying database instance is skewed, a querying pattern focusing on the skewed portion of data is more likely to require more query samples than the querying pattern that only queries the uniformly distributed data portion.

High-level idea. We are aware of many papers [22, 27, 28] in learning theory which based on certain assumptions, derives the general sample complexity for various ML models (*e.g.,* Bayesian networks, neural networks) directly from the model. However, it is unclear how to distribute the general sample complexity into different query clusters that are formed in an online fashion. Therefore, to account for the sample complexity, instead of the direct estimation from the model, we utilize the query losses during training as the proxy for the cluster sample complexity. On a high level, if the queries in a cluster have a large mean training loss, it is likely that the cluster size does not match the actual sample complexity of the cluster. As such, we need to increase the size of the cluster. We call this strategy Loss-Weighted Buffer Population (LWP).

Implementation. We implement this idea by revisiting the Op2 (RS_ASSIGN_CENTER) and Op3 (RS_ASSIGN_NONCENTER) of CBP. For Op2, if there are only center instances in the buffer, rather than removing centers by reservoir sampling (*i.e.,* uniform sampling), we make the cluster with a higher mean loss have a smaller probability to be removed, which, in essence, is equivalent to increasing the size of the cluster. Specifically, we remove a center (including the new center) with the probability *inversely weighted* by the center loss. Next, for Op3, instead of removing a non-center instance from the largest center, again we consider removing an instance from a cluster with the probability *inversely weighted* by the cluster mean loss. This way, clusters with sufficient queries (small loss) are more likely to reduce in size compared to those lacking training queries (large loss), when the buffer is filled.

Discussion. The benefits of the loss-based proxy of LWP are at least two-fold. First, LWP is model-agnostic, making it compatible with any ML model. In contrast, sample complexity derived from learning theory is specific to a fixed model family or even a fixed parameter setting [4], requiring new effort for a new model. Second, LWP only requires further storing of training losses of queries which can be directly obtained during training without additional computation. This requires very little storage budget and almost no extra computational budget.

7 EXPERIMENTAL EVALUATION

In this section, we aim at answering the following research questions regarding ShiftHandler:

1. How does the proposed replay buffer perform for various shifting workloads compared to other replaying methods? (§ 7.2) Can the shift detector detect workload shifts? (§ 7.3) Do the conclusions hold with varying workload shift ratios? (§ 7.4)
2. How does ODMedoids compare to online DP-Means? (§ 7.5)
3. Is ShiftHandler efficient and practical? (§ 7.7)

7.1 Experimental Setup

We evaluate ShiftHandler on three core components in learned DB. Note that ShiftHandler is applicable to other learned workload-driven DB tasks as well, such as CPU usage prediction [60].

- *Learned Cardinality Estimators*, which predict the join sizes of queries. We integrate ShiftHandler with MSCN [37], a well-known learned query-driven cardinality estimator.
- *Learned Cost Estimators*, which estimate the cost for a chosen query plan. We integrated ShiftHandler with Tree-CNN [51], which encodes queries on the plan level.
- *Learned Query Optimizers*. To further investigate how ShiftHandler improves the query performance of learned query optimizers, we integrate ShiftHandler with a learned bandit query optimizer, Bao. Bao uses a learned cost/latency model (Tree-CNN) for estimating the posterior distribution of model weights θ and then choosing the best arm. Hence the estimation quality of the cost model is the key to the success of Bao.

Baselines. We evaluate several baseline strategies for replaying: (1) Retraining from the latest s queries (Latest). For a fair comparison, we set the s in Latest to match the size of the current batch plus the query size m of the replay buffer; (2) Retraining from all previous queries (ALL); (3) Buffer population via reservoir sampling (RS) which essentially uniformly samples from the workload. We also evaluate (4) Postgres (PG) for reference. For the proposed replay buffer, we evaluate both (5) CBP and (6) LWP. Note that we exclude Postgres from the experiment of learned cost estimators because of the fact that the notion of "cost" in Postgres (which is in an arbitrary *unit of computation step*) differs from that in learned cost estimators (which is the query latency in wall-clock time), making them incomparable. Instead, it is more reasonable to compare the qualities of query plans produced by them, which are shown in the experiment over learned query optimizers.

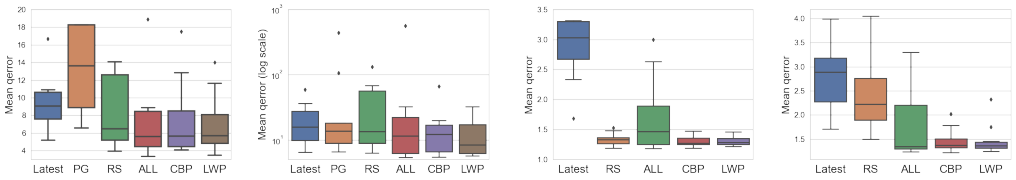
We also experiment with retraining without a replay buffer (i.e., a buffer of $m = 0$). The result is drastically worse than retraining with a replay buffer. We do not report the less informative result for space, but it proves the usefulness of maintaining a replay buffer.

Datasets and Query Classes. We conduct the evaluations on the IMDb [43] database, a complex real-world dataset on films and television programs. We consider the full schema of IMDb (21 relations), with the JOB benchmark [43]. We also use the recently proposed DSB benchmark [20] to conduct the most critical experiment of learned query optimizers. DSB enhances the TPC-DS benchmark with complex data distribution and challenging query classes. We populate a 50GB DSB database with the default physical design configuration. Table 1 shows the statistics of the DB instances and their classes. We run IMDb-related experiments on Ubuntu 20.04.5 LTS with Intel Core 7-9700K CPU @ 3.60GHz and 16GB RAM, and run DSB-related experiments on an AWS EC2 c5.9xlarge node.

DB Instance	# Tables	Size (G)	# Classes	# Avg Joins
IMDb	22	1.1	33	8
DSB	25	50	15	6.5

Table 1. Statistics of data and classes of DB instances

Workload Simulation. We first generate base queries from each class. For learned cardinality estimators and learned cost estimators, we directly use the 100K queries with up to two joins in the



(a) Left: Shifting workloads; Right: S&I workloads (b) Left: Shifting workloads; Right: S&I workloads

Fig. 4. Performance of cardinality estimation (left) and cost estimation (right) on two types of shifting workloads (e.g., Shifting workloads and S$I workloads)

MSCN project [37]. For learned query optimizers, we use 32^2 classes (with different join graphs) in the JOB benchmark [43] to generate queries from various classes (200 queries for each) on IMDb. For DSB, we also generate 200 queries per class with Gaussian distribution (variance 1). These queries serve as the "query pool" from which we simulate shifting and imbalanced workloads.

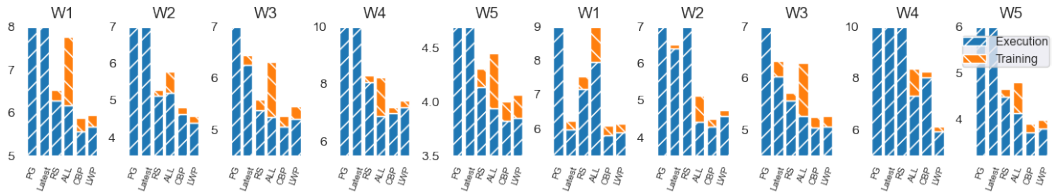
We then simulate two types of workloads as the testbed, shifting workloads as well as shifting and imbalanced (S&I) workloads. The S&I workloads are to further assess the robustness of retraining approaches to workload imbalance. More specifically, a workload contains n_c (3 for learned cardinality/cost estimators and 4 for learned query optimizers) randomly picked query classes. Each workload starts with a set of burn-in queries randomly sampled from the query pool, in which the number of queries is balanced across classes for shifting workloads but becomes imbalanced for S&I workloads. The burn-in queries equip the learned components with basic prior knowledge. Afterward, we proceed to simulate workload shifts — we randomly sample a set of n_t (1000/200/100 for learned cardinality estimators/cost estimators/learned query optimizers) queries for each class and concatenate them for both types of workloads. In other words, after the burn-in process, each workload shifts for every n_t query for both workload types.

Metrics and Evaluation Protocol. The performance is evaluated after the burn-in process. For learned cardinality/cost estimators, we use Q-Error (smaller is better) as the metric which has been widely used in previous work [37, 51, 74]. At the end of every n_t query of class T, we retrain the learned components with different retraining approaches and evaluate the performance on the test queries of different classes (100 per class). This is to evaluate the model's learning performance on all seen query classes. We exclude class T because we observe no difference in the performance on T since it forms the current batch. We report the mean aggregated results of all retraining points. Since the ultimate goal of learned components is to reduce the query running time, for learned query optimizers, we use the total running time (query execution time plus model training time) of queries as the evaluation metric. For all the experiments, the trade-off parameter ψ is set to 0.5, and the replay buffer limit m is set to 3% $\sim$ 5% of the entire workload size. For CBP and LWP, λ in ODMedoids is chosen from $\{0.01, 0.1, 1, 10, 100\}$.

7.2 Performance and Findings

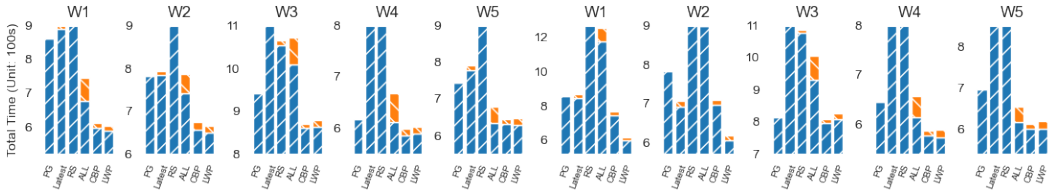
Figure 4 shows the experimental results of learned cardinality/cost estimators. Both experiments are conducted on ten randomly generated workloads. We also plot the performance of learned query optimizers (Bao) with different replay strategies in Figures 5 and 6. Note that we truncate the running/training time for several approaches for visual simplicity, wherever the upper stacked bar is not shown: *e.g.*, the execution time of Latest on W3 of Figure 6a is 2317s. But we remark that

[2]We exclude class 23, for which the average query running time is more than 5 minutes.



(a) Shifting workloads (b) S&I workloads

Fig. 5. Query execution and model training time of Bao with replaying strategies on IMDb workloads



(a) Shifting workloads (b) S&I workloads

Fig. 6. Query execution and model training time of Bao with replaying strategies on DSB workloads

model training time is not negligible, especially for ALL. Overall, we draw a few important and interesting scientific findings from the results.

F1. Learned database systems could suffer from workload shifts without a carefully designed replay buffer for retraining, which could be exacerbated by workload imbalance. This can be concluded from the performances of Latest and RS(Figure 4). Specifically, both Latest and RS perform the worst among the baselines excluding Postgres. The only exception is RS on the shifting workloads of cost estimation, which is still slightly worse than CBP and LWP. The exception is likely because the query plans used in the cost estimation experiments are generated only with the default Postgres setting without using any query hints so that the resulting plans do not contain as wide varieties as those in the leaned query optimizers experiment. Therefore, RS could capture the uncomplicated workload. This conjecture is again confirmed in the experiment of learned query optimizers (Figures 5, 6) — since the query plans generated from multiple query hints are far more complicated, Latest and RS are not able to represent the corresponding workload characteristic. This makes Bao easily produce poor plans for both workload types on both database instances. *Particularly,* Latest*'s and* RS*'s overall performances are even worse than* Postgres *on DSB, exemplifying the importance of strategically forming the replay buffer for retraining in learned query optimizers.*

F2. Even neglecting retraining efficiency, retraining from all previous queries does not necessarily achieve the best model performance. One may expect ALL can achieve the best model performance since it uses all the previous queries for retraining. However, perhaps surprisingly, this is not always the case. In the cost estimation experiment (Figure 4b), ALL is far from the best approach. Interestingly, ALL is even worse than RS for shifting workloads. We observe 2 out of 10 shifting workloads on which ALL performs significantly poorer than RS. We suspect that this is because the ML model is overparameterized for the classes in the two workloads, which causes the *double descent* phenomenon [3, 23] — As the model complexity increases, the test risk first follows the standard U-shaped curve in classical learning theory until the point where the training data is fit perfectly (interpolation threshold), but after the point, the risk begins to descend again. The phenomenon endorses our solution of *not using all previous queries.*

For the learned query optimizers experiment, while ALL is much better than RS and Latest for shifting workloads, it becomes drastically less stable for S&I workloads, especially on the DSB benchmark. This again demonstrates that learned DB systems could suffer from workload imbalance. Moreover, even though ALL achieves better model performance (less query execution time) than RS and Latest for the IMDb S&I workloads, the total running time is still unfavorable due to the considerable training time. For example, on W3 ALL is indeed worse than RS and Latest in terms of overall running time. *This shows that* ALL *makes an unsuitable trade-off between model quality and training overhead for learned DB components.*

F3. CBP **can mitigate the effect of workload shifts on learned database systems with minimal retraining overhead.** It is easy to see CBP performs better than baselines in the experiments of learned cardinality/cost estimators (Figure 4). Notably, the superiority of CBP is pronounced in the learned query optimizers experiment, which is more critical than the other two learned components. For the 10 shifting workloads of the two database instances (Figures 5, 6), CBP consistently achieves the best overall performance, without significant regression in query execution time, even compared to ALL. To be more specific, CBP reduces the total running time of Latest by 42% and 35% on IMDb and DSB, respectively. It is also worth noting that even only considering the query execution time, the reduction ratios of CBP on ALL are 5% on IMDb and 8% on DSB. The two ratios become 16% and 14% if including the training time. To further validate CBP's superiority over ALL under *heavy* workloads, we also compare CBP to ALL with larger workload sizes. We would like to note that the superiority becomes much more significant with a heavier workload. *Therefore, we conclude that* CBP *strikes the best trade-off between model performance and training overhead for handling workload shifts in learned query optimizers.*

F4. CBP **is more robust to workload imbalance compared to baselines.** This is witnessed from the observation that the improvement of CBP on baselines tends to increase when moving from shifting workloads to S&I workloads. For example, one can easily see from Figure 4b that the improvement of CBP over RS is remarkably larger from shifting workloads to S&I workloads. A similar trend is also shown in Figure 4a, noting that the right figure of S&I workloads is in log scale because a few outliers appear due to workload imbalance. For the learned query optimizers experiment (Figures 5, 6), as an example to illustrate, the total running time reduction ratio of CBP on RS increases from 34% on DSB shifting workloads to 39% on DSB S&I workloads. Additionally, while CBP's overall model performance is still better than ALL for S&I workloads on both database instances (despite that ALL outperforms CBP on the W4 of IMDb S&I workloads), we observe that the improvement is much more significant on DSB over IMDb. This is likely because the DSB classes are more challenging than the IMDb classes, with more correlated and skewed underlying data distribution.

F5. LWP **is more robust to outliers than** CBP**.** Both CBP's and LWP's performances are very close on shifting workloads in all three evaluated database components. However, on more challenging S&I workloads where outliers (*e.g.,* very inaccurate cardinality estimates, catastrophic query plans) are more likely to occur, LWP is more robust than CBP. For the cardinality estimation experiment, LWP indeed has a significant improvement over CBP in terms of tail performance (Q-error 66 *vs.* 32) for S&I workloads, which is not obvious to see from Figure 4a in log scale. For the learned query optimizers experiment on IMDb S&I workloads (Figure 5), LWP can prevent outliers on W4 which, as mentioned before, cause CBP's execution time longer than ALL. Interestingly, we also observe that LWP is slightly worse than CBP on W2 and W5. We suspect *it is because* LWP *trades-off a majority of "easy" queries with small losses for "difficult" queries with large losses on which* LWP *focuses more.* Similar trends can be observed from DSB S&I workloads (Figure 6) — LWP significantly outperforms CBP on workloads W1 and W2, while slightly underperforming CBP on workloads W3 and W5.

7.3 Accuracy of Shift Detection

Input	TP	FP	TN	FN	Recall	Precision
Query Features	1000	1	999	0	1	≈ 1
Labels	855	38	962	145	0.86	0.96
Query Features+Labels	1000	39	961	0	1	0.96

Table 2. Performance of workload shift detection.

To test the accuracy of the shift detector in ShiftHandler, we conduct experiments over the IMDb benchmark with a confidence level 0.05. We first randomly generate 1000 shift workload (of 100 queries) pairs. Two workloads of a shift pair are from different classes. Moreover, to check against false positives, we generate 1000 no-shift workload pairs of the same size, but workloads of the same pair share the same class. We evaluate three variants of inputs for the shift detector — 1) original query features 2) labels, and 3) query features+labels based on which we initiate two distribution monitors separately and predict a workload shift if either triggers. We use the query encoding of MSCN.

Table 2 shows the True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN), recall, and precision of shift detection. We observe that 1) the shift detector with query features as input can achieve very accurate (near-perfect) performance in both recall and precision and 2) query features are more critical than labels in detecting workload shifts in learned DB systems. Thus we recommend using query features (or possibly query features+labels if emphasizing high recall).

7.4 Varying the Workload Shift Ratio

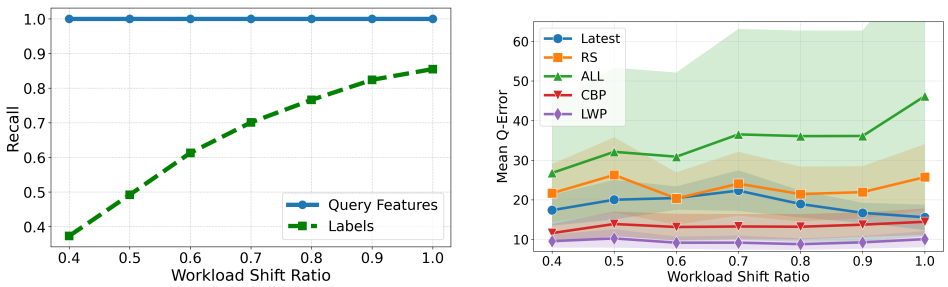


Fig. 7. Performance of shift detector (left) and replay buffer (right) with varying workload shift ratios.

Realistic workloads may contain a wide variety. In this subsection, we conduct additional experiments to validate that ShiftHandler can maintain its performance under various settings of workload shift. Therefore, we repeat some experiments in previous subsections (shift detection and cardinality estimation over S&I workloads) but with the workload shift ratio (r%) varying from 0.5 to 1, to evaluate the effects of varying the workload shift ratio on the shift detector and the replay buffer, respectively. Specifically, at each shift point, only r% queries are shifted to the new query class, and other queries' classes are unchanged. This way we simulate the scenario where the workload shifts gradually.

Figure 7 shows the results. It is clear that the recommended input (Query Features) achieves consistently near-perfect accuracy with varying r. For the replay buffer, we observe that CBP and

LWP exceed baselines and LWP outperforms CBP under different r. Thus we conclude that our major scientific findings still hold with varying workload shift ratios.

7.5 DP-Medoids *vs.* DP-Means

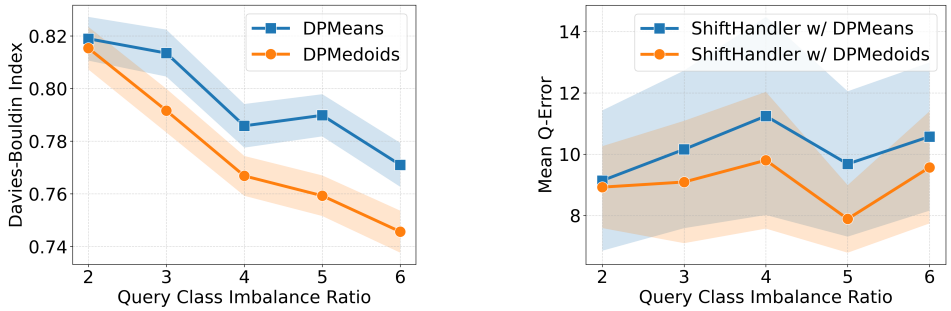
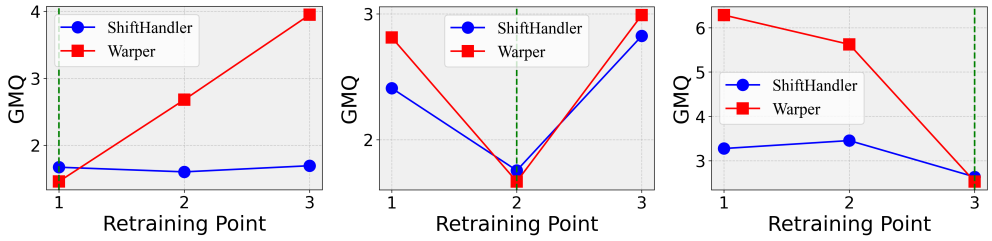


Fig. 8. Robustness of DP-Medoids and DP-Means against various levels of workload imbalance. Left: Davies-Bouldin Index (DBI) performance (smaller is better). Right: Mean Q-Error performance of integrating into ShiftHandler.

Recall that in § 5.2, we argue that better robustness to outliers is a key rationale for choosing K-Medoids over K-Means for our online replay buffer. We conduct ablation studies to justify the argument.

DP-Medoids vs DP-Means. First, we show that DP-Medoids can inherit this benefit by comparing it to DP-Means [40], built on top of K-Means. We generate query workloads with various levels of query Class Imbalance Ratio (CIR) [65], which measures the ratio of the sample size of the largest majority class and that of the smallest minority class. It provides a straightforward way to gauge workload imbalance (a larger CIR value indicates a larger workload imbalance). To measure the clustering quality, we compute the well-known Davies-Bouldin Index (DBI) [17] of the clustering results from DPMedoids and DPMeans. Davies-Bouldin Index measures the average similarity between each cluster and its most similar cluster, and a lower value suggests better clustering. For each CIR, we randomly generate and run 100 query workloads of size 100 and compute the values of Davies-Bouldin Index. For example, if CIR is 4, the sample size distribution could be 15, 25, and 60 for classes 1, 2, and 3. Figure 8 (left) illustrates that their clustering qualities are close with a small workload imbalance (*e.g.,* CIR is 2). However, with increasing levels of workload imbalance (*i.e.,* higher CIR), DP-Medoids shows increasing benefit over DP-Means. This demonstrates the superior robustness of DP-Medoids over DP-Means against workload imbalance.

Can DP-Medoids Improve ShiftHandler**'s Performance?** Next, we investigate whether DP-Medoids' robustness can translate into improved predictive performance. We implement a DP-Means version of ShiftHandler and compare its cardinality estimation performance on S&I workloads with the DP-Medoids one. We vary the CIR values for the S&I workloads and compute the mean Q-Error. Fitting K-Means into ShiftHandler requires that we use the arithmetic centers in DP-Means to compute distances when assigning queries to clusters. For each cluster, we pick the query nearest to the arithmetic center as the "center query" for the subsequent buffer management steps. Figure 8(right) establishes that DP-Medoids's superiority over DP-Means can indeed translate into a better performance of ShiftHandler when handling workload imbalance.



(a) Query Class 1 (b) Query Class 2 (c) Query Class 3

Fig. 9. Comparison of ShiftHandler and Warper.

7.6 Comparison with Warper

Warper [44] is a framework that accelerates query-driven model adaptation to workload shifts, specifically for cardinality estimation. Warper uses a Generative Adversarial Network [29] (GAN) to generate training queries of the new query class and annotate them at each retraining point. Warper's goal is solely fast adaptation; ShiftHandler seeks both fast adaptation (using new queries) and long-term memory (maintaining a replay buffer).

We implement Warper with the default hyperparameters and model architectures [44]. With the experimental setup of Section 7.1, we compare ShiftHandler with Warper in terms of the prediction errors of the learned cardinality estimator (MSCN) on shifting workloads and use the geometric mean of q-error (GMQ, following the Warper paper) as the metric to evaluate both approaches. We simulate workload shifts at three points, each towards a query class. After each retraining point, we break down the results of three query classes in the three subfigures in Figure 9. Green dashed lines indicate the points at which the workload shifts to the query class represented in that subfigure: e.g., in subfigure (a), at retraining point 1, the workload shifts to class 1 (with subsequent shifts to classes 2 and 3 at points 2 and 3, respectively).

Warper improves the performance of query classes to which the workload shifts (green lines). However, compared to ShiftHandler, Warper largely degrades for other (old) query classes in the workload. This is because ShiftHandler and Warper have different emphases. Overall, ShiftHandler achieves a better trade-off between adapting to new workloads and memorizing old workloads.

7.7 Efficiency and Practicality

Next, we discuss the efficiency of ShiftHandler. For the shift detector, the average detection time is only 0.08s per pair of workload size 100. The average detection time is 0.33s, even after increasing the workload size to 1K. The most computationally expensive part of the replay buffer is finding the nearest cluster center for each query. Thus we deploy a k-d tree [9] to accelerate the process. The resulting processing time is highly efficient. In the learned query optimizers experiment on IMDb, the total processing time of the replay buffer for each retraining is only 33ms on average, less than 0.5% of the average model training time (7.7s). We observe the same trend on DSB. To further assess the scalability of the replay buffer, we simulate 10K queries with random query encodings of 1K dimensions, much higher than common practice in learned DB components. The total processing time for the entire set is only 14s (1.4ms per query). Note that the efficiency result is very close for CBP or LWP. We also assess the offline DP-Medoids approach (Alg 1), where even a *single* iteration for processing the last query takes 8s, which is highly impractical compared to our replay buffer.

Based on the efficiency studies, we conclude that ShiftHandler *is highly scalable, with negligible overhead compared to model training time or query execution time.* Moreover, it is easy to implement

and tune in practice — besides the significance level (often set to 0.05) and query batch size for the shift detector, there are only two parameters ψ and λ for the replay buffer.

8 RELATED WORK

The notion of adapting to observed workloads dates back to the earliest days of database query processing and led to a line of work in adaptive query processing [18]. However, the focus of much of that work was learning from partial query evaluation, and adapting execution to match. A key approach has been to leverage knowledge from subexpressions computed during execution, and generalizing to others — for histograms [1, 13, 46], query expression statistics [12] and adjustments to correlated predicates [52].

Over the past decade, approaches based on ML have shown great promise. These **learned workload-driven database systems** can learn or improve an ML model for a variety of database components, also by using the execution log of a query workload. For example, the first works on cardinality estimation were [5–7]. More recently, there is active work on workload-aware cardinality estimators [37, 70, 72], cost estimators [62, 64, 76, 77], and query optimizers [33, 49, 50]. Beyond query optimization, query workloads are being used for building learned workload-driven indexes [21, 54, 67], knob tuning [45, 68, 75] and database generation [73]. A recent related work is DDUp [41] which focuses on data shifts for *data-driven* models. Although targeting different kinds of models, ShiftHandler could use techniques from DDUp to handle data shifts, as discussed in Section 3. Other recent work [44, 55] studies workload shifts in learned DBs, but from different perspectives — [55] reduces the need for retraining by tweaking the MSCN model; Warper [44] generates queries for fast adaptation. They are both specifically designed for cardinality estimation. However, our focus is to provide learned DB components (which could extend beyond cardinality estimation) with a general replay buffer-based framework that can achieve both plasticity and long-term memory without changing the model. While our evaluation in Section 7.6 demonstrated that, compared to Warper, ShiftHandler strikes a better balance in adapting to new workloads and keeping long-term memory for cardinality estimation, it is worth noting that ShiftHandler is compatible with and can be combined with both [44, 55].

9 CONCLUSION AND FUTURE WORK

We proposed a general replay buffer-based framework, ShiftHandler, an important step toward learning workload-driven DB systems over shifting workloads. We have provided a near-optimal online workload clustering algorithm and two novel strategies for buffer management. We also empirically demonstrated that ShiftHandler mitigates the effect of workload shifts on learned DB systems, even with workload imbalance.

For future work, we believe there is more to do in exploring additional online clustering algorithms and buffer management strategies. A possible alternative to DP-Medoids would be (online) Gaussian mixture models which can fit more complex data patterns than DP-Medoids and help with robustness to outliers. We are also interested in bringing information about data distributions and constraints (as in traditional query optimization) into the learned optimizer's reasoning process — these could help bound the error during workload shifts.

ACKNOWLEDGMENTS

The authors would like to thank our anonymous reviewers for their many insightful comments, which were beneficial to the revision of the paper. We gratefully acknowledge the support of the National Science Foundation, via grant award III-1910108, for portions of this work.

REFERENCES

[1] Ashraf Aboulnaga and Surajit Chaudhuri. 1999. Self-tuning histograms: Building histograms without looking at data. *ACM SIGMOD Record* 28, 2 (1999), 181–192.

[2] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. Join Synopses for Approximate Query Answering. 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, 275–286. https://doi.org/10.1145/304182.304207

[3] Ben Adlam and Jeffrey Pennington. 2020. Understanding double descent requires a fine-grained bias-variance decomposition. *Advances in neural information processing systems* 33 (2020), 11022–11032.

[4] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. 2019. Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems* 32 (2019).

[5] Christos Anagnostopoulos and Peter Triantafillou. 2015. Learning set cardinality in distance nearest neighbours. In *2015 IEEE international conference on data mining*. IEEE, 691–696.

[6] Christos Anagnostopoulos and Peter Triantafillou. 2015. Learning to accurately count with query-driven predictive analytics. In *2015 IEEE international conference on big data (big data)*. IEEE, 14–23.

[7] Christos Anagnostopoulos and Peter Triantafillou. 2017. Query-driven learning for predictive analytics of data subspace cardinality. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 11, 4 (2017), 1–46.

[8] Charles E Antoniak. 1974. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The annals of statistics* (1974), 1152–1174.

[9] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[10] Allan Borodin and Ran El-Yaniv. 2005. *Online computation and competitive analysis*. cambridge university press.

[11] Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. 2011. Streaming k-means on well-clusterable data. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 26–40.

[12] Nicolas Bruno and Surajit Chaudhuri. 2002. Exploiting statistics on query expressions for optimization. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 263–274.

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

[14] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *Database Theory—ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings 8*. Springer, 316–330.

[15] Pierluigi Crescenzi. 1997. A short guide to approximation preserving reductions. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*. IEEE, 262–273.

[16] Sanjoy Dasgupta. 2008. *The hardness of k-means clustering*. Department of Computer Science and Engineering, University of California.

[17] David L Davies and Donald W Bouldin. 1979. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence* 2 (1979), 224–227.

[18] Amol Deshpande, Zachary Ives, Vijayshankar Raman, et al. 2007. Adaptive query processing. *Foundations and Trends® in Databases* 1, 1 (2007), 1–140.

[19] Or Dinari and Oren Freifeld. 2022. Revisiting DP-Means: Fast Scalable Algorithms via Parallelism and Delayed Cluster Creation. In *The 38th Conference on Uncertainty in Artificial Intelligence*.

[20] 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.

[21] Haowen Dong, Chengliang Chai, Yuyu Luo, Jiabin Liu, Jianhua Feng, and Chaoqun Zhan. 2022. Rw-tree: A learned workload-aware framework for R-tree construction. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2073–2085.

[22] Simon S Du, Yining Wang, Xiyu Zhai, Sivaraman Balakrishnan, Russ R Salakhutdinov, and Aarti Singh. 2018. How many samples are needed to estimate a convolutional neural network? *Advances in Neural Information Processing Systems* 31 (2018).

[23] Stéphane d'Ascoli, Maria Refinetti, Giulio Biroli, and Florent Krzakala. 2020. Double trouble in double descent: Bias and variance (s) in the lazy regime. In *International Conference on Machine Learning*. PMLR, 2280–2290.

[24] Tongtong Fang, Nan Lu, Gang Niu, and Masashi Sugiyama. 2020. Rethinking importance weighting for deep learning under distribution shift. *Advances in neural information processing systems* 33 (2020), 11996–12007.

[25] Dimitris Fotakis. 2008. On the competitive ratio for online facility location. *Algorithmica* 50, 1 (2008), 1–57.

[26] Dimitris Fotakis. 2011. Online and incremental algorithms for facility location. *ACM SIGACT News* 42, 1 (2011), 97–131.

[27] Nir Friedman and Zohar Yakhini. 2013. On the sample complexity of learning Bayesian networks. *arXiv preprint arXiv:1302.3579* (2013).

[28] Noah Golowich, Alexander Rakhlin, and Ohad Shamir. 2018. Size-independent sample complexity of neural networks. In *Conference On Learning Theory*. PMLR, 297–299.

[29] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *NIPS* 27 (2014).

[30] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *VLDB* 13, 7, 992–1005.

[31] Marc Holze and Norbert Ritter. 2007. Towards workload shift detection and prediction for autonomic databases. In *Proceedings of the ACM first Ph. D. workshop in CIKM*. 109–116.

[32] Yannis E Ioannidis and Stavros Christodoulakis. 1991. On the propagation of errors in the size of join results. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of data*. 268–277.

[33] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Bertty Contreras-Rojas, Rodrigo Pardo-Meza, Anis Troudi, and Sanjay Chawla. 2020. ML-based cross-platform query optimization. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1489–1500.

[34] Oded Kariv and S Louis Hakimi. 1979. An algorithmic approach to network location problems. I: The p-centers. *SIAM journal on applied mathematics* 37, 3 (1979), 513–538.

[35] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. 2018. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

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

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

[38] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. 2021. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*. PMLR, 5637–5664.

[39] Andrey Kolmogorov. 1933. Sulla determinazione empirica di una lgge di distribuzione. *Inst. Ital. Attuari, Giorn.* 4 (1933), 83–91.

[40] Brian Kulis and Michael I Jordan. 2011. Revisiting k-means: New algorithms via Bayesian nonparametrics. *arXiv preprint arXiv:1111.0352* (2011).

[41] Meghdad Kurmanji and Peter Triantafillou. 2023. Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[42] Erich Leo Lehmann and EL Lehmann. 1986. *Testing statistical hypotheses*. Vol. 2. Springer.

[43] 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.

[44] 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.

[45] 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.

[46] Lipyeow Lim, Min Wang, and Jeffrey Scott Vitter. 2003. SASH: A self-adaptive histogram set for dynamically changing workloads. In *Proceedings 2003 VLDB Conference*. Elsevier, 369–380.

[47] Lin Ma, Bailu Ding, Sudipto Das, and Adith Swaminathan. 2020. Active Learning for ML Enhanced Database Systems. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 175–191. https://doi.org/10.1145/3318464.3389768

[48] J MacQueen. 1965. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium on Math., Stat., and Prob.* 281.

[49] 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.

[50] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A learned query optimizer. In *VLDB*.

[51] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-structured deep neural network models for query performance prediction. *PVLDB* (2019).

[52] Volker Markl, Guy M Lohman, and Vijayshankar Raman. 2003. LEO: An autonomic query optimizer for DB2. *IBM Systems Journal* 42, 1 (2003), 98–106.

[53] Adam Meyerson. 2001. Online facility location. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 426–431.

[54] 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.
[55] 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.
[56] Shigeyuki Odashima, Miwa Ueki, and Naoyuki Sawasaki. 2016. A split-merge DP-means algorithm to avoid local minima. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 63–78.
[57] Peter Orbanz and Yee Whye Teh. 2010. Bayesian Nonparametric Models. *Encyclopedia of machine learning* 1 (2010).
[58] Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert systems with applications* 36, 2 (2009), 3336–3341.
[59] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. 2019. Failing loudly: An empirical study of methods for detecting dataset shift. *Advances in Neural Information Processing Systems* 32 (2019).
[60] Gaurav Saxena, Mohammad Rahman, Naresh Chainani, Chunbin Lin, George Caragea, Fahim Chowdhury, Ryan Marcus, Tim Kraska, Ippokratis Pandis, and Balakrishnan Narayanaswamy. 2023. Auto-WLM: Machine learning enhanced workload management in Amazon Redshift. In *Companion of the 2023 International Conference on Management of Data*. 225–237.
[61] Michael Shindler, Alex Wong, and Adam Meyerson. 2011. Fast and accurate k-means for large datasets. *Advances in neural information processing systems* 24 (2011).
[62] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. 2020. Cost models for big data query processing: Learning, retrofitting, and our findings. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 99–113.
[63] Nikolai V Smirnov. 1939. On the estimation of the discrepancy between empirical curves of distribution for two independent samples. *Bull. Math. Univ. Moscou* 2, 2 (1939), 3–14.
[64] Ji Sun and Guoliang Li. 2019. An end-to-end learning-based cost estimator. *VLDB* 13, 3 (2019), 307–319.
[65] Fadi Thabtah, Suhel Hammoud, Firuz Kamalov, and Amanda Gonsalves. 2020. Data imbalance in classification: Experimental evaluation. *Information Sciences* 513 (2020), 429–441.
[66] Anbupalam Thalamuthu, Indranil Mukhopadhyay, Xiaojing Zheng, and George C Tseng. 2006. Evaluation and comparison of gene clustering methods in microarray analysis. *Bioinformatics* 22, 19 (2006), 2405–2412.
[67] Kostas Tzoumas, Man Lung Yiu, and Christian S Jensen. 2009. Workload-aware indexing of continuously moving objects. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1186–1197.
[68] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.
[69] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* 11, 1 (1985), 37–57.
[70] Chenggang Wu, Alekh Jindal, Saeed Amizadeh, Hiren Patel, Wangchao Le, Shi Qiao, and Sriram Rao. 2018. Towards a learning optimizer for shared clouds. *VLDB* 12, 3 (2018), 210–222.
[71] CF Jeff Wu. 1983. On the convergence properties of the EM algorithm. *The Annals of statistics* (1983), 95–103.
[72] Peizhi Wu and Gao Cong. 2021. A unified deep model of learning from both data and queries for cardinality estimation. In *Proceedings of the 2021 International Conference on Management of Data*. 2009–2022.
[73] 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.
[74] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2021. NeuroCard: One Cardinality Estimator for All Tables. *PVLDB* (2021).
[75] 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.
[76] Johan Kok Zhi Kang, Sien Yi Tan, Feng Cheng, Shixuan Sun, and Bingsheng He. 2021. Efficient deep learning pipelines for accurate cost estimations over large scale query workload. In *Proceedings of the 2021 International Conference on Management of Data*. 1014–1022.
[77] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query performance prediction for concurrent queries using graph embedding. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1416–1428.

Received July 2024; revised October 2024; accepted November 2024