Warper: **Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts**

Beibin Li
University of Washington
Seattle, WA, USA
Microsoft
Redmond, WA, USA

Yao Lu
Microsoft
Redmond, WA, USA

Srikanth Kandula
Microsoft
Redmond, WA, USA

ABSTRACT

Recent learned cardinality estimation (CE) models are vulnerable when query predicates or the underlying datasets drift from what the models were trained upon. We propose a system Warper that accelerates model adaptation to drifts; Warper generates additional queries when limited examples are available from the new workload and carefully picks which queries to use to update the CE model. We show that Warper can be used to adapt different CE models including ones that support queries over single tables and join expressions. Experiments with different drifts suggest that Warper has a small computational cost and adapts much faster compared to state-of-the-art solutions. We also show that faster model adaptation improves query performance by shortening the period for which imperfect query plans are picked by a query optimizer due to incorrect cardinality estimates.

CCS CONCEPTS

• **Information systems → Database query processing**; • **Theory of computation → Database query processing and optimization (theory)**; *Online learning algorithms.*

KEYWORDS

Cardinality estimation, database optimization, data shift, adaptation

ACM Reference Format:
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 (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3526179

1 INTRODUCTION

We consider learned cardinality estimation (CE) models in query optimizers and other applications. Examples include Learned Models (LM) [10] and MSCN [25]. Such models train over a corpus of (predicate, cardinality) pairs for each relation and aim to estimate the cardinality for a given, possibly unseen, predicate. These models show promising accuracy on predicates that are similar to those used in training; however, changes to the underlying data or predicates are known to cause sizable drops in accuracy [10, 21, 25, 48].

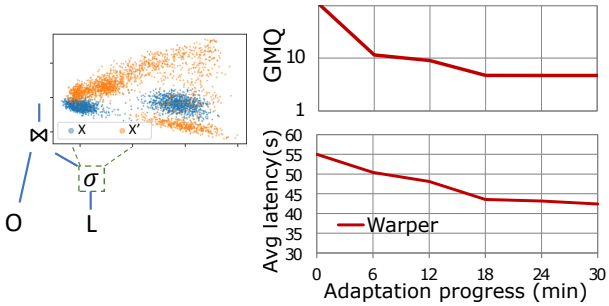

Figure 1: Visualizing the effect of workload drift on the performance of a simple query using tables from TPC-H [1]; the predicate shown is drawn from distribution X which changes to X';[1] on the right, we see how the GMQ[2] of cardinality estimates from [10] and the query latency vary at different adaptation steps when new queries arrive from X'.

To respond to changes in data or predicate workload, prior solutions suggest updating the models by re-training or fine-tuning using newly drawn (predicate, cardinality) pairs [10, 25]. Since these models require training sets of several thousands of queries, re-obtaining the updated training corpus is costly and has high latency. If only the data changes, the ground-truth cardinality labels will have to be recomputed [9, 25]. Worse, when the predicate workload changes, the models may have to wait until enough new queries appear. The net result is that model adaptation is slow and expensive; meanwhile, poor cardinality estimates lead to imperfect plan choices and degrade the query performance.

Example. To concretize these concerns, consider the simple select-project-join query in Figure 1; here L and O denote the Lineitem and Orders tables from TPC-H [1] and we consider different choices for the predicate on L. During the CE model construction, the training predicates are drawn from the distribution shown in blue (marked X) while newly arriving predicates to test the CE model are drawn from the distribution in orange (marked X').[1] On the right, we show the GMQ[2] of cardinality estimates for LM [10] as well as the actual query latency when this query executed with TPC-H inputs at a scale factor of 10 on a desktop-class machine. By improving the model accuracy, Warper improves the query performance in a sizable way; as the figure on the right shows, adapting to workload

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526179

[1] This is a 2-d visualization of multi-variate predicates; more details are in §2.
[2] GMQ is the geometric mean of q-error, a well-used metric for CE, defined in §4.1.

drift reduces cardinality estimation errors by up to 3× and the query latency improves by 31%[3].

An ideal adaptation technique should satisfy the following requirements. First, the costs to adapt models should be small, especially relative to the benefits. Next, different scenarios require different adaptations. For example, when only the dataset changes, i.e., the query workload is stable, a key concern is which ground truth cardinality labels to re-obtain to keep costs small. When the predicate workload changes and only limited examples are available, a key concern is how to generate more examples that mimic the new workload. Finally, when or how often should the model be updated, and how to use examples that have possibly inaccurate ground truth and are possibly no longer representative of the newly arriving predicates? With Warper, we take a first stab at these aspects.

One part of Warper is inspired by recent advances in Generative Adversarial Networks (GANs)[13, 19, 40]: a generator aims to synthesize examples that are indistinguishable from the observed new predicates, and an adversarial discriminator aims to distinguish the synthetic queries from actually observed queries; we train both in an adversarial game. Warper also has a picker that predicts which input queries are more valuable to update the CE model and reduces costs by only annotating (fetching up-to-date ground truth cardinality labels for) those queries.

Warper takes as input the newly arriving predicates, database update statistics and a CE model that has been trained from workload history. It identifies the nature of the ongoing drift, learns to generate additional queries when necessary, and updates the CE model. The output of Warper is an improved CE model that is more accurate given the ongoing drift. Notably, Warper is agnostic to and uses the underlying CE model as a black box (e.g., the model can be LM [10] or MSCN [25]); Warper does not modify the model's structure and hyper-parameters and applies directly to the predicates that the model can support; improvements accrue from generating and labeling useful examples so that the CE model adapts quickly.

Adapting previously trained models to different drifts (e.g., covariant and concept drifts) has been studied in the machine learning literature [41]; a few different ideas have been used to synthesize new examples, such as using clustering heuristics or by adding noise to the observed examples [7, 16, 38, 47]. We find these augmentations manual, task-specific, expensive, and inefficient. Also, in AI applications, new examples can be used for training without ground truth [50] or ground truth acquisition can be delayed [41], done only when needed [39], or be computed using existing labels [28]. We find that adapting CE models is challenging since generating synthetic predicates intertwines in a complex way with acquiring or updating the ground truth labels.

We are unaware of any prior work that effectively adapts learned database models to data and workload drifts. In our experiments on a typical database server without GPU, Warper speeds up model adaptation by many times over solutions from the ML literature while using <1% extra CPU utilization (details in §4). Warper incorporates different learned components to handle different drifts and generates synthetic queries only when necessary. Finally, we show that faster adaptation translates to query performance gains by injecting cardinality estimates into a production query optimizer.

The **contributions** of this paper are as follows:

- We demonstrate how data and workload drifts can affect ML models that learn over workload history. Drifts often lead to significant accuracy and query performance deterioration, and prior solutions may require many examples to adapt.
- We propose Warper, a system that collaborates with an existing CE model in a non-invasive manner and accelerates model adaptation to changes in the data and workload.
- Experiments show that Warper offers a useful cost-speedup tradeoff; at a small cost, Warper adapts many-fold faster than fine-tuning and other baselines. Warper also generalizes to different CE models and handles a variety of individual and continuous drifts in the data and workload.

2 BACKGROUND AND MOTIVATION

Learned database components have attracted great attention recently. Recent works have demonstrated promising results in learned indexes [27], filters [33] and query optimization [36]. Warper adapts cardinality estimation (CE) models [10, 25] to different drifts; here, we describe relevant work and the scope of Warper.

Scope of the underlying CE models. Warper is designed to be agnostic to the underlying CE model; that is, we will show that Warper can be used to adapt several different CE models including LM [10] and MSCN [25]; we also show results for variants of LM that use gradient boosted trees [14], multi-layer perceptrons (MLP) [18] and support vector machines [6]. The LM model covers predicates of the following kind on individual tables while the MSCN model covers key-foreign key join conditions in addition to predicates of the following kind on each table: **SELECT** `count(*)` **FROM** T **WHERE** $\bigwedge_i l_i \leq \text{Col}_i \leq u_i$, where T is a table and columns can have values of date, numeric or categorical types. Note that equality and one-sided range predicates belong in the above class: for equality, we set $l_i = u_i$; for one-sided ranges, we set $l_i = \min(\text{Col}_i)$ or $u_i = \max(\text{Col}_i)$, and for columns that are not in the predicate, we set $(l_i, u_i) = (\min \text{Col}_i, \max \text{Col}_i)$. Some generalizations also follow directly, e.g., using multiple calls for disjunctions.

Recent CE models can be classified into those that learn from workload history [10, 25] and those that learn directly from the dataset [21, 48]. While both kinds of models lose accuracy upon changes, in this paper, we focus on adapting the workload-driven models because they can be adapted by re-training or fine-tuning with an updated set of queries. On the other hand, data-driven models lack a common adaptation pattern; some must be rebuilt from scratch when data changes [48] while others use model-specific updates that do not generalize [21].

Workload drift vs. data drift. We denote a learned CE model as $\mathtt{M}_{X,D,y}$ if it was trained using the predicate workload X on table D and with cardinality labels y. We focus on these three aspects that are agnostic to model internals because the workloads and the table can change independently during a drift. The labels will change when either of them changes. The workloads and labels are inputs to model training and re-training. We already saw an instance of workload drift in Figure 1 wherein X changed. By *data drift*, we

[3]The GMQ is about 7 after convergence vs 19 before adaptation. Imperfect CE lead to excessive buffer spills; more details are in §4.2.

refer to changes to the underlying dataset, e.g., inserts, appends, deletes, or updates to rows:[4] here, the ground-truth cardinality labels y become inaccurate, and the previously trained model loses accuracy. For example, the cardinality estimation error of the LM model [10] on the Power dataset [8] increases by 2× when 20% of the rows are appended and over 55× when 100% of the rows are updated. In absolute terms, the model goes from being somewhat usable with no updates (a GMQ of 1.8 indicates that cardinality is under-estimated by 44% or over-estimated by 80% on average) to rather unusable when 20% of the data changes (a GMQ of 3.6 which indicates an average under-estimation of 72% or over-estimation of 260%). In practice, we find that workload and data drifts happen continuously and simultaneously.

Visualizing workload drift. We briefly sketch our method to visualize workload drifts which we have used in Figure 1. Range predicates with low and high values on d columns can be represented as a $2d$-dim vector. Since most relations have many columns (high d), we convert predicates into fewer dimensions using Principal Component Analysis (PCA) [46]; specifically, we compute eigenvectors by running singular value decomposition (SVD) over all predicates, select the two highest weighted eigenvectors, and represent each predicate using its projection on those eigenvectors. The resultant 2-d plots allow us to qualitatively compare the drift between workloads.

Space of adaptation solutions and where Warper **fits in.** Broadly speaking, adaptation techniques solve these problems: (1) determine when the model must change, (2) acquire new training examples, e.g., predicates and cardinality labels, that more closely resemble future workload, (3) figure out how to combine new examples with the previous training examples that are possibly inaccurate and (4) update the existing model using these examples.

There are two popular ways to update models. By *fine-tuning* we refer to training the model for a few more epochs with the updated workload; model training is typically iterative (e.g., using SGD to train neural networks) and fine-tuning has been shown to be practically useful [18, 49]. By *re-training* we refer to training the model (with the same structure and hyperparameters) from scratch over the updated workload. The computation cost of the model update depends on the complexity of the model; we find that the latency to re-train ranges from minutes for the neural network-based LM models [10] to tens of minutes for the more complex MSCN model [25].

Acquiring up-to-date training examples includes identifying new queries and obtaining up-to-date ground-truth cardinality labels for them. The latter typically requires querying the DBMS; doing so can be expensive even if implemented efficiently, e.g., batching predicates into a single evaluation tree and executing many predicates in one query still scans the underlying table at least once. Some prior works suggest using samples [9]; since predicates can have a wide range of selectivities, one must use a bag of samples of different types and sizes, which in turn increases the complexity to maintain samples. Also, sampling-induced errors can affect model quality. Instead, to acquire new training examples on DBMSs that have a high and continuous rate of queries, Warper carefully

[4]We defer supporting schema changes to future work.

Notation	Meaning
γ	# of annotated queries needed for a robust model.
$\mathbb{I}_{\texttt{train}}$	The original training workload (i.e., a set of queries) that was used to build the CE model.
n_i	#iterations to update the Warper modules in each invocation.
n_g, n_p, n_a	# of examples to generate by $\mathbb{G}$, pick by $\mathbb{P}$ and annotate by $\mathbb{A}$.
n_t	# of arrived queries from the new workload.

Table 1: Notations used in this paper.

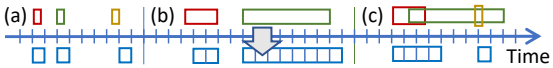

Figure 2: Top: different colored boxes show different kinds of drifts. Bottom: we show boxes whenever Warper adapts the CE model; note that Warper periodically evaluates if the model needs to be adapted and handles different kinds of drifts in a unified manner.

and judiciously picks which queries to use to update the model. In the converse case, when a DBMS has only a slow rate of queries, rather than wait until enough new queries arrive, Warper generates synthetic queries that mimic the actual ones to speed up adaptation.

There is sizable prior work in ML literature on generating realistic synthetic examples [38, 47] including heuristics, e.g., add noise to the observed queries [43] and posterior generative models [15]. With Warper, we will show that using Generative Adversarial Networks (GANs) leads to better quality. GANs have shown promising results in synthesizing image, text, and audio examples [20, 24, 30] which in turn have helped to train object detectors and image segmenters [22, 23]. Warper occupies a different point in the design space; unlike AI applications that may use sparse or delayed annotations, obtaining cardinality labels incurs an immediate cost before the new queries can be used to update the model.

Problem formulation. Given a CE model $\mathbb{M}$ that is previously trained from workload history and a few example queries from the new predicate workload (along with their ground truth cardinality if available), we consider adapting the CE model to data and workload drifts quickly using few resources. Table 1 shows the notations used in this paper. The **goals** of our adaptation method are:

- Accelerate adaptation, i.e., achieve the same accuracy quickly,
- use only a small amount of additional resources,
- support a wide variety of CE models and,
- cover a wide scope of drifts with a unified solution.

Notably, for workload drifts, adapting CE models requires an *adequate* number of labeled queries.[5] We consider different scenarios based on whether incoming queries are inadequate and whether ground truth can be computed for all queries. Drifts can also be complex; they can happen jointly (e.g., a drift that affects both data and workload) and change frequently. Figure 2 shows complex drifts including (a) short-lived drifts, (b) persistent or continuous drifts, and (c) combinations of different types of drifts.

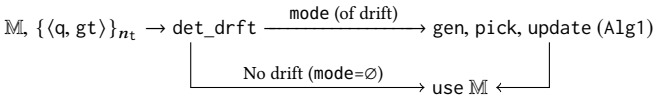

Figure 3: Given queries with cardinality labels {⟨q,gt⟩} from the new workload, Warper **uses different strategies for adaptation.**

[5]We show in §3.1 how to determine if actual queries are adequate to update a model.

	Drift	Mitigations in Warper			Note
		gen?	pick?	update?	
c1	Data	×	✓	✓	Unchanged workload, slow labeling.
c2	Wkld	✓	✓	✓	Inadequate incoming queries.
c3	Wkld	×	✓	✓	Slow labeling, can happen with c2.
c4	Wkld	×	×	✓	Adequate queries with labels.

Table 2: Individual data and workload (wkld) drifts and adaptation strategies. Complex cases are combinations of individual drifts.

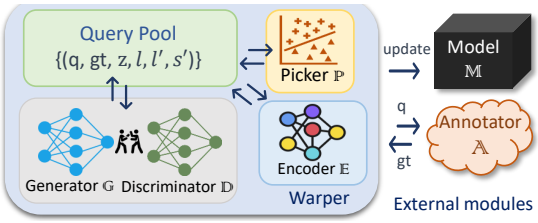

Figure 4: Architecture of the Warper system.

3 DESIGN OF WARPER

Warper adapts to drifts periodically. The inputs for Warper are a CE model $\mathbb{M}$, database telemetry and queries with cardinality labels (if available) $\{\langle \texttt{q,gt} \rangle\}$ from the new workload. As shown in Figure 3, at each period, where zero, one or more kinds of drifts may happen, Warper detects the type of drift that is underway and adapts using one or more of the following strategies:

- gen: When there are inadequate new queries to update $\mathbb{M}$, Warper synthesizes additional queries that mimic the new workload.
- pick: When acquiring ground-truth cardinality labels cannot keep up, Warper conserves resource usage by picking useful queries to annotate.
- update: Warper updates[6] $\mathbb{M}$ using labeled queries.

Table 2 shows how these strategies combine for some example drift types. (1) Cases c2 and c3 correspond to workload drifts, wherein new queries are inadequate (in c2) and annotation is slow (in both); Warper generates additional queries for c2 and picks the queries to annotate in both cases. (2) In case c4, wherein an adequate number of labeled queries are available, Warper directly updates $\mathbb{M}$.

System architecture, shown in Figure 4, has four key components.

- *The query pool* maintains tuples of (q, gt, z, l, l', s') wherein q is a predicate with ground truth cardinality gt and l denotes the source of the predicate – a prior training workload (l = train), the new workload (l = new) or synthesized (l = gen). Other parameters are defined next.
- The GAN which emits synthetic queries consists of a *generator* $\mathbb{G}$ and the discriminator $\mathbb{D}$. l' is the predicted source of the predicate from $\mathbb{D}$ and s' is the confidence score for the prediction.
- *The picker* $\mathbb{P}$ borrows ideas from active learning [41] to perform a weighted sampling over the given set of predicates.
- The *encoder* $\mathbb{E}$ embeds predicates q into a different space z which is used by the other components.
- Note the two external modules here: (1) the *CE model* $\mathbb{M}$, short for $\texttt{M}_{X,D,y}$, is the previously trained model that Warper aims to improve; Warper needs not know the structure of $\mathbb{M}$. (2) The *annotator* $\mathbb{A}$ computes ground truth for query predicates and can be a DBMS query or custom code.

Next, we describe our approach end-to-end which detects and adapts to drifts in §3.1. We discuss the key components of Warper in §3.2 and §3.3. Finally, we discuss the system robustness in §3.4.

3.1 Detect, identify, and adapt to drifts

Detect drifts: Prior learned CE models suggest updating periodically or whenever enough new queries are available [10]. As shown in Figure 3, Warper uses evaluation feedback from database telemetry to identify a potential drift when the evaluation error of the CE model on the newly arriving queries exceeds the error observed during training by more than a threshold ($\delta_m > \pi$). Thus, Warper adapts only when model accuracy has degraded. We discuss further details, caveats, and corner cases in §3.4. If no drift is detected, Warper simply uses the current model (mode=∅ in Figure 3). This det_drft trigger is simple and easy to implement efficiently.

Identify drift modes: Each det_drft call also characterizes the drift that is underway as one of the cases in Table 2 which we call the mode flag; thus, mode can indicate a data drift c1 and/or a workload drift {c2,c3,c4}. More than one kind of drift may be detected at a time as we discuss next.

Data drifts. In data drifts, the cardinality labels for all queries (including those from the previous training set $\mathbb{I}_{\texttt{train}}$) may be outdated. To identify data drift from D to D', we use different measures, including (1) counting the fraction of rows that are new or have changed since the model was last trained, and (2) measuring the change in ground truth cardinality for a few canary predicates. A data drift sets the c1 bit in the mode flag. For data drifts, we must re-obtain cardinality labels.

Workload drifts. We leverage a symmetric form of the discrete Jensen-Shannon Divergence [35] to measure the drift in workload. Specifically, if A and B denote the newly arriving predicate set and the predicates that the model was trained upon previously, we apply PCA[7] to reduce predicates to k-dims. Next, we quantize each dimension into m bins; thus, each predicate becomes a value in $[0, k^m)$. Third, we compute k^m-bucket histograms H_A, H_B respectively over the predicates in each workload where each histogram bucket stores normalized frequencies. Finally, we compute a symmetric discrete KL-divergence [29] measure.[8] The $\delta_{\texttt{js}}$ metric is a value in $[0, 1]$ with 0 indicating identical distributions.

We identify different kinds of workload drifts as follows: (1) c2 denotes the case when newly arrived queries are inadequate; that is, the number of new queries available (n_t) is below γ, the number of annotated queries necessary to train a robust CE model. We estimate γ offline based on the training size at which the accuracy of $\mathbb{M}$ stabilizes and tune γ, online, based on how the accuracy of $\mathbb{M}$ stabilizes during adaptations. (2) c3 denotes the case when the number of queries with ground truth labels is inadequate, i.e., $n_a < \gamma$; this can happen because computing the labels is too slow or too expensive or when execution feedback contains cardinality

[6] Fine-tune or re-train, depending on $\mathbb{M}$; see §3.2.

[7] similar to the visualization strategy described in §2, each is a matrix where each row represents a predicate.

[8] Let $M = 1/2(A + B)$, then $\delta_{\texttt{js}}(A, B) = 0.5 * (\texttt{KL}(A, M) + \texttt{KL}(B, M))$ where $\texttt{KL}(F, G) = \sum_x H_F(x)(\log H_F(x) - \log H_G(x))$. To prevent numeric error, we add a small constant to each $H(x)$.

Algorithm 1: Warper procedures in an invocation.

Input : Newly arrived {⟨q, gt⟩}, drift mode from det_drft, $\mathbb{M}$, $\mathbb{G}$, $\mathbb{D}$, $\mathbb{E}$ from the previous invocation.

1 pool.append({⟨q, gt, l=new⟩});
2 **if** mode contains c1, c2 or c3: // *mitigate drifts using* gen *and* pick.
3 **if** mode contains c2: //*generate synth. queries and update GAN.*
4 **while** n_i--: // *up to n_i iterations, see Table 1.*
5 q_{gen} ← pool.gen($\mathbb{G}$, $\mathbb{E}$, n_s); // *gen. n_s synth. queries.*
6 pool.update_MultiTask($\mathbb{G}$, $\mathbb{E}$, $\mathbb{D}$, q_{gen}); //*see §3.3.*
7 pool.append({⟨q_{gen}, l=gen⟩});
8 **else:** pool.update_AutoEncoder($\mathbb{G}$, $\mathbb{E}$); //*see §3.3.*
9 anno(pool.pick($\mathbb{E}$, mode, n_p)); //*update gt for n_p queries.*
10 $\mathbb{M}$ ← update($\mathbb{M}$, pool); // *update the underlying CE model $\mathbb{M}$.*

Output : Updated CE model $\mathbb{M}$ and internal models $\mathbb{G}$, $\mathbb{D}$, $\mathbb{E}$.

estimates for some but not all of the expressions that a query optimizer will consider during planning [34]. (3) c4 is the converse case when both queries and ground truth are adequate. As we discuss in §3.4, Warper is rather robust to inaccuracies in estimating γ since the drift type identification repeats in each period. Warper also uses a form of early stopping to reduce unnecessary resource usage due to false positives in drift detection. Note that higher CE error for newly arriving predicates may indicate that the workload has drifted or may just be due to outlier predicates that belong in the previous distribution. To balance quick reaction to drifts with reducing the resources consumed by false positives, Warper adapts the error threshold (π from earlier in §3.1) over time.

Adapting to individual drifts. Alg. 1 further fleshes out the actions in Figure 3. We will give a brief overview here and will discuss in more detail in subsequent paragraphs. Warper injects newly arrived predicates into the query pool (line#1). Lines#4–#6 update the generator and discriminator if synthetic queries are needed (c2). Note that the GAN models and the Encoder adapt on-the-fly during the update_ calls in lines#6 and #8. Line#9 picks the queries to use for training and annotation. Finally, line#10 updates the CE model using predicates and labels from the pool. We describe details of the modular calls, e.g., gen(), in §3.2 and §3.3. It is easy to see that many calls can be parallelized.[9]

Adapting to complex drifts. When multiple kinds of drifts happen at a time, i.e., mode = c1|c2 or c2|c3 and so on, Warper combines different mitigation strategies. Note that Alg. 1 presents different combination of strategies based on the value of the mode flag. For continuous drifts, as discussed earlier, Warper repeats the drift detection and adapts periodically (e.g., in each x-tick of Figure 2) using the same adaptation strategy from Alg. 1.

3.2 Using Warper Components

We describe in detail the design and implementation of individual modules used in Warper as shown in Figure 4 and Alg. 1.

The query pool is an in-memory data structure that maintains queries and associated content as shown in Figure 4. Warper initializes the query pool using the original training workload ($\mathbb{I}_{train}$). That is, for each (q, gt) tuple in $\mathbb{I}_{train}$, Warper creates a record in the query pool with l = train and empty values for z, l', s'. During each adaptation step, i.e., each call to Alg. 1, newly arriving queries along with their cardinality labels (if available) are injected into the query pool with l = new. When necessary, synthetic queries (q_{gen} in Alg. 1) generated using the GAN are also added to the query pool with l = gen and gt=-1. During the course of adaptation, the different components Warper update fields in the pool, e.g., the annotator $\mathbb{A}$ updates gt, the encoder $\mathbb{E}$ updates z and the discriminator $\mathbb{D}$ updates the predicted label and confidence score (l', s').

The CE Model $\mathbb{M}$ can be any function that emits a cardinality for a given query predicate: q → $\mathbb{M}$ → card which can update() itself using additional labeled predicates (Line#10 in Alg. 1). Warper aims to improve the CE model without needing to know the model design.[10] The appropriate update process differs across models; for example, neural networks (NNs) are iteratively trained and can be fine-tuned but tree-based models usually need to be re-trained from scratch. Featurization is also model specific; for example in LM [10], each query is featurized as the vector $\{low_1, .., low_d, high_1, .., high_d\}$ where low_i and $high_i$ denote the low and high range checks for the $i-th$ column of a table. MSCN [25] on the other hand uses a featurization that allows for predicates over multiple tables; Warper supports whichever featurization is used by the model $\mathbb{M}$.

Table 3 lists the structures of learned models used in Warper.

Layer	Encoder $\mathbb{E}$	Generator $\mathbb{G}$	Discriminator $\mathbb{D}$
1	Fully Conn. 128	Fully Conn. 128	Fully Conn. 3
2	Leaky ReLU activation	Leaky ReLU activation	
3	Fully Conn. 128	Fully Conn. 128	
4	Leaky ReLU activation	Leaky ReLU activation	
5	Fully Conn. 128	Fully Conn. 128	
6	Leaky ReLU activation	Leaky ReLU activation	
7	Fully Conn. $\|z\|$	Fully Conn. m	

Table 3: Specifications of the learned Warper modules. $|z|$: the embedding size. m: input size to $\mathbb{M}$.

The Encoder $\mathbb{E}$ transforms a predicate into a compact embedding vector as representation:

$$q \rightarrow \mathbb{E} \rightarrow z.$$

Recall that Warper is agnostic to the CE model $\mathbb{M}$ and each may use a different featurization, Warper leverages learned encoder to decouple internal components (i.e., $\mathbb{G}, \mathbb{D}, \mathbb{P}$) from the featurization used by $\mathbb{M}$. We found that doing so improves the performance of Warper. In our implementation, embed() uses the ground truth labels as an additional input (beyond q) whenever they are available and up-to-date. The encoder is learned on the fly and embeddings are updated in each invocation of Alg. 1.

The Generator $\mathbb{G}$ synthesizes new query predicates using the predicate embeddings in the pool:

$$z + \epsilon \rightarrow \mathbb{G} \rightarrow q_{gen},$$

where $\epsilon \sim \mathbb{N}(0, \sigma^2)$ is a random Gaussian noise where σ is the standard deviation of z, the embeddings of the previously seen predicates. $\mathbb{G}$ is a simple NN as shown in Table 3. Whereas prior generative methods use random seeds ϵ as input [19], we find that generating from $z + \epsilon$ is more likely to resemble the new workload. $\mathbb{G}$ is trained together with the discriminator $\mathbb{D}$ to formulate a GAN.

[9] A multi-threaded version of this algorithm is in our tech report [2].

[10] By model design, we refer to model structures (e.g., neural network vs. decision forest etc.) or hyperparameters.

The Discriminator $\mathbb{D}$ is another NN that takes a predicate embedding as input and predicts whether a predicate resembles the training, new or synthetically generated workload:

$$\texttt{z} \rightarrow \mathbb{D} \rightarrow l' \in \{\texttt{gen}, \texttt{new}, \texttt{train}\}, s'.$$

As we will show in §3.3, $\mathbb{G}$ and $\mathbb{D}$ update in each Warper invocation; Warper uses and updates $\mathbb{D}$ only in workload drift `c2` with inadequate incoming queries. Table 3 demonstrates the model parameters of $\mathbb{G}$ and $\mathbb{D}$. Our goal, here, is to use simple models; we find that these models suffice to handle the wide variety of drifts that we experiment with. We recommend considering different models or increasing the model sizes as we show in §4.3 to tune these models to handle complex future drifts; we also defer more careful model selection and hyper-parameter tuning to future work.

The Picker $\mathbb{P}$ sub-selects a specified number of queries from the pool which are more useful to update the model $\mathbb{M}$ and thus reduces the annotation cost. As shown in Alg. 1, the picker has two distinct use-cases:

- For drift `c2`, wherein $\mathbb{D}$ generates synthetic queries, $\mathbb{P}$ uses a weighted sampling with replacement over those queries ($l' =$ `gen`) based on their confidence score s'; synthetic queries that more closely resemble the newly arriving queries are picked.
- For drift `c1` and `c3`, the picker performs a sampling stratified by the CE error. Specifically, we first cluster all records in the pool which have cardinality labels into k buckets based on their evaluation error over estimates from $\mathbb{M}$. Next, for each new query without cardinality labels, we assign it to one of the buckets based on k-nearest neighbor using its embedding z. Finally, we pick records from different buckets with replacement to make a stratified sample. Doing so picks predicates to annotate from across a wide range of CE errors; we find this leads to a better model update using a smaller annotation cost.

3.3 Training Warper Components

There are three components in Warper that are learned: the Encoder $\mathbb{E}$, the Generator $\mathbb{G}$ and the Discriminator $\mathbb{D}$. We formulate training and updating them as an end-to-end, multi-task learning in `update_AutoEncoder()` and `update_MultiTask()` depending on the drift type and if generating new queries is needed.

`update_AutoEncoder`**:** For drift types that do not require generating new queries (`c1` and `c3`, see Table 2), we train and update an auto-encoder that consists of the encoder $\mathbb{E}$ and the generator $\mathbb{G}$:

$$\texttt{q}, \texttt{gt} \rightarrow \mathbb{E} \rightarrow z \rightarrow \mathbb{G} \rightarrow \hat{\texttt{q}}.$$

The training goal here is to minimize the reconstruction (L1) loss between the input query predicate and the recovered one:

$$\mathcal{L}_{\texttt{AE}} = |\texttt{q} - \hat{\texttt{q}}|. \tag{1}$$

To compute $\mathcal{L}_{\texttt{AE}}$. we use all records from the pool no matter if ground truth labels are available[11]. Therefore, the encoder and the generator can generalize to all three cases (i.e., queries from the generator, the training, or the new predicate workload). Once loss functions are computed, training the auto-encoder and updating the weights of $\mathbb{E}, \mathbb{G}$ follow the standard back-propagation approach [18].

[11] `gt=-1` when ground truth labels are not immediately available

`update_MultiTask`**:** When synthetic queries are needed, the task here is to update the generator $\mathbb{G}$ together with the discriminator $\mathbb{D}$ in a GAN fashion [19]. For each iteration in Line#4-6, $\mathbb{G}$ generates a set of query predicates $q_{\texttt{gen}}$ as described earlier and $\mathbb{D}$ distinguishes them for belonging to $\{\texttt{gen}, \texttt{new}, \texttt{train}\}$:

$$z + \epsilon \rightarrow \mathbb{G} \rightarrow \texttt{q}_{\texttt{gen}} \rightarrow \mathbb{E} \rightarrow z' \rightarrow \mathbb{D} \rightarrow l'.$$

The generator loss is defined by:

$$\mathcal{L}_{\texttt{gen}} = \texttt{CrossEntropy}(l', l = \texttt{new}).$$

The goal here is to generate queries as close to the new workload as possible and the discriminator classifies the generated query correctly as `new`. A $\texttt{CrossEntropy}(p, q) = -\sum_{x \in X} p(x) \log q(x)$ is a standard loss function for training classifiers [18].

On the other hand, the discriminator is trained by classifying the generated queries as well as the existing ones from the pool (Line#6 where inputs can have all three cases of l):

$$\texttt{q}, \texttt{gt} \rightarrow \mathbb{E} \rightarrow z \rightarrow \mathbb{D} \rightarrow l_d.$$

The discriminator loss is defined by:

$$\mathcal{L}_{\texttt{discr}} = \texttt{CrossEntropy}(l, l_d),$$

so that the discriminator correctly identifies the origin (i.e., training, newly arrived, or synthetic) of the query predicates. Finally, the GAN loss is a combination of the generator and the discriminator losses:

$$\mathcal{L}_{\texttt{GAN}} = \mathcal{L}_{\texttt{gen}} + \mathcal{L}_{\texttt{discr}}. \tag{2}$$

The training simultaneously optimizes $\mathcal{L}_{\texttt{gen}}$ and $\mathcal{L}_{\texttt{discr}}$ so that $\mathbb{G}$ and $\mathbb{D}$ play against each other and formulate a GAN – minimizing $\mathcal{L}_{\texttt{gen}}$ so that $\mathbb{G}$ generates predicates that can be classified as $l' =$ `new` despite that actual labels for generated queries are $l =$ `gen`; minimizing $\mathcal{L}_{\texttt{discr}}$ so that $\mathbb{D}$ recognizes the generated predicates. Unlike a classic GAN training in which a binary label of {`new`, `gen`} is used, here we use a three-class label {`gen`, `new`, `train`}, since `train` is not presented in classic GANs and can be sufficiently different from `new`.

3.4 Robustness in Warper

We note a few design considerations, caveats and corner cases here.

Early stop in Warper. We use the accuracy gain of $\mathbb{M}$ after each adaptation step as the stopping criteria; once the gain is less than a small threshold, `det_drft` for the next invocation uses a larger π (§3.1), so that Warper directly uses the previous CE model unless a larger drift is observed (e.g., due to a new drift). This strategy saves computations, because possible improvements are already minor at such a moment. Early stopping also adds robustness to inaccurate γ and drift type identification; details follow.

Robustness and fallback options in Warper are as follows.

Drift detections: (1) False negatives - in a drift when `det_drft` does not trigger (i.e., the accuracy gap of the drift is small or even negative), Warper uses the existing CE model and no action is needed, because empirically there is small accuracy degrade already. (2) False positives, i.e., when there is a large accuracy gap but no drift, is practically impossible.

Drift type identifications. For data drifts, the underlying database system should provide reliable signals. Still, in the event of faulty

telemetry: (1) False positives - the bottom line is to re-compute ground truth and Warper falls back to prior learned CE solutions [10], which has no negative impact on the model accuracy. (2) False negatives - this is the same as FN in drift detection.

For workload drifts, we know if `gt` is available from the new workload and the exact rate of computing `gt`; hence `c3`, which explicitly checks for low or no `gt`, cannot be confused from `c2` or `c4`. Therefore, with an overestimated γ, `det_drft` yields `c2` instead of `c4` and all the Warper modules are used. In such a scenario, there are adequate queries and cardinality labels; the Warper modules converge quickly which often triggers an early stop in practice. When γ is underestimated, `det_drft` yields `c4` instead of `c2`. Warper falls back to updating $\mathbb{M}$ directly and is no worse than fine-tuning. We use a simple heuristic here to tune γ in runtime - when `det_drift` yields `c4` while the accuracy improves slowly, Warper uses a larger γ since the aforehand signal can indicate an underestimated γ. Nevertheless, Warper is robust to γ and has a bottom line for no worse than re-training or fine-tuning.

Adaptation intervals and outliers from the new workload. Since Warper runs independently to n_t – the number of incoming queries available for each invocation, it is also robust to the adaptation intervals chosen by the users. Indeed, when n_t is small, using a mean error estimation for $\mathbb{M}$ brings in uncertainty for a real workload drift or outliers in the incoming queries, which in turn result in inaccurate control decisions. However, by the early stop and other mechanisms discussed above, Warper corrects itself when arrived queries are adequate.

Choosing a large adaptation interval, as shown by the yellow box in Figure 2 (a), may cause delayed drift detection. Since `det_drft` has a small overhead, we use frequent Warper invocations in practice. Besides, Figure 2 (c) shows an example of early stop.

3.5 Implementation Details

We implement a prototype of Warper in Python; $\mathbb{G}, \mathbb{D}$ and $\mathbb{E}$ are implemented in sklearn and PyTorch with a learning rate of 1e-3 and half-decay after every 10 epochs. The query pool is an in-memory array, and the annotator $\mathbb{A}$ is in C++. Our experiments were performed on a typical database server with a 12-core Intel CPU at 2.9GHz and without GPU. We use $n_i = 100$ in `task2` and use an early stopping when the loss converges.

When $\mathbb{I}_{\texttt{train}}$ that is used to train the original $\mathbb{M}$ is available, the generator $\mathbb{G}$ and the encoder $\mathbb{E}$ are pre-trained offline using `task1` and the queries from $\mathbb{I}_{\texttt{train}}$. In such a manner, the initial weights of $\mathbb{E}$ and $\mathbb{G}$ are determined; in our experiments, we find that this pre-training strategy speedups the GAN convergence in each Warper invocation. Such pre-training incurs a one-time cost that is similar to training the LM [10] model offline.

4 EXPERIMENTS

We evaluate Warper against state-of-the-art adaptation solutions on a wide scope of drifts. Recall from §3.1 that Warper adapts periodically to complex and continuous drifts. We also show that faster adaptation translates to query plan improvements. The goals of this section are as follows.

`T1` (§4.1) When adapting different CE models to different types of *individual* drifts, Warper outperforms various baselines, has a small CPU cost (e.g., about 1% on a typical database server) and adapts faster, reaching a similar accuracy in shorter time.

`T2` (§4.2) We show end-to-end gains using three select-project-join queries on TPC-H under *continuous* drifts. Faster adaptation leads to a shorter period of query latency regression since the query optimizer starts picking better plans sooner.

`T3` (§4.3) We tease apart the usefulness of different components in Warper using ablation studies; we also analyze the sensitivity and costs of using different parameter choices.

Table Name	Cols Real	Cols Cat.	Rows n	Distinct count Min/Medium/Max
Higgs	28	0	11M	3/6.7K/290K
PRSA	16	2	430K	5/645/35K
Poker	0	11	1M	4/10/13

Table 4: Datasets for evaluation in experiments. Cat.: categorical.

	Method to generate {`low`, `high`} predicates for column C.
`w1`	Draw from $r(C)$ uniformly at random.
`w2`	Draw from a logarithmic transform of $r(C)$.
`w3`	Equal to a sampled row plus a random width in $r(C)$
`w4`	Equal to min(C), max(C) from a sample of k rows.
`w5`	Equal to a stratified sample row by frequency plus a random width in $r(C)$

Table 5: Different methods to generate workloads. $r(C)$ denote the value range in column C.

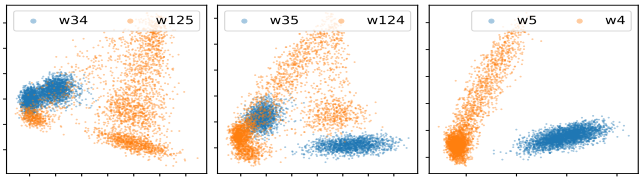

Figure 5: Visualizing some workloads on PRSA in our experiments.

4.1 Can Warper help to mitigate drifts?

Datasets. In our experiments, we use the datasets shown in Table 4. These datasets have a rich variety in terms of row and column counts, column types, distinctness, and have been used in prior CE solutions [8, 10]. Further, we leverage the IMDB dataset [31] to evaluate adapting a join CE model.

Workloads. We have not identified public datasets with realistic drifts. As shown in Table 5, we use five methods to generate query predicates which have been applied in [3, 10, 37] to evaluate CE solutions or are simple modifications to existing methods. For example, LM [10] used a mixture of `w1+3` in their paper. Using a large set of workload distributions enables us to better evaluate the generalizability of various adaptation methods. Figure 5 demonstrates some workload visualizations using the PCA method in §2.

Metrics. We measure the following key metrics:

Accuracy: Let $g, \widehat{g}$ be the estimated and actual cardinalities, for each predicate, we measure the q-error: $q_\theta(g, \widehat{g}) = \max(\frac{\max(g,\theta)}{\max(\widehat{g},\theta)}, \frac{\max(\widehat{g},\theta)}{\max(g,\theta)})$. This is a widely used metric [10, 25, 48] so that lower q-error indicates better accuracy and 1 is perfect accuracy; To prevent numeric error, we use $\theta = 10$ to follow [10]. For each test relation, we measure the geometric mean of q_θ over all predicates (GMQ) also to follow prior work [10, 25, 48].

Test period and query arrival rate. How fast an adaptation solution mitigates a drift crucially depends on the time period of the drift

and the query arrival rate being tested. Hence, in our experiments, we leverage a fixed test time period of 30 mins. As we will show by the cost analyses in §4.3, the CPU cost incurred by Warper in a time unit is formulated by $c_{\texttt{gt}} + C$ where the first term is the cost to annotate queries and the second term is a constant to update models, etc.; the relative adaptation speedups remain *the same* with different arrival rates of new queries, as long as Warper can keep up with the compute, i.e., average CPU usage is small during the test period or before the model adapts. We use one test query arrival per five seconds in the experiments (unless otherwise specified) and will report costs at different arrival rates.

Computational overhead. We measure the cost of Warper to build and apply different learned components and report the latency aggregated in one thread. For Warper and various baselines, the costs consist of the time to generate and annotate additional queries (if necessary) and update the CE model. Warper additionally requires updating its learned components. We report average CPU utilization on a desktop-grade database server with 12 cores.

Relative adaptation speedup, agnostic to the test period and query arrival rate, evaluates the effectiveness of a model adaptation solution. For a CE model, accuracy improvements with different numbers of training examples are nearly monotonic, as shown in Figure 1. Let α, β, respectively, be the GMQ before and after the drift; we define $\Delta(A, \lambda)$ as the number of queries required for method A to reach a GMQ at most $\beta + \lambda(\alpha - \beta)$.

We use a relative speedup $\Delta(FT, \lambda)/\Delta(A, \lambda)$ by comparing the numbers of queries required from the new workloads for method A relative to that for fine-tuning (FT). For example, $\alpha = 3.0$ and $\beta = 2.0$, fine-tuning requires 100 queries to reach a GMQ of 2.5, while method A requires 50; hence $\Delta(FT, 0.5)/\Delta(A, 0.5) = 100/50 = 2$, indicating that A has a 2× speedup to reach half of the possible improvement compared with FT. In the rest of this paper, we denote $\Delta_{.5}$, $\Delta_{.8}$ and Δ_1 for short of $\Delta(FT, \lambda)/\Delta(A, \lambda)$ where $\lambda \in [.5, .8, 1]$.

Drift metrics: To measure the severity of a drift, we leverage *blind* and *intrinsic* metrics from the active learning literature [16].

$\delta_{\texttt{m}}$: Agnostic to the underlying data or workload drifts, the blind metric $\delta_{\texttt{m}}$ captures the accuracy gap between an unmodified model and the model trained exclusively on the new data and workload. Such metric is used in `det_drft` (§3.1) and early stop §3.4.

$\delta_{\texttt{js}}$: We leverage the discrete Jensen-Shannon Divergence [35] as described in §3.1 to measure the intrinsic distance between two workloads. We use $k = 10$ and $m = 3$ in our experiments.

Baselines. We describe the baselines used in our experiments:

Fine-Tuning (FT) existing models is a well-adopted method to mitigate drifts [10, 19]; we use fine-tuning as the baseline to measure adaptation efficiency for other solutions. The baseline re-trains (*RT*) the CE model whenever it cannot fine-tune.

Data Augmentation (AUG): Given limited arrival of new queries, AUG is one step further than FT to update the CE model using augmented examples. Data augmentation uses some pre-defined heuristics or rules and has been widely used in many AI domains to improve model generalization, e.g., adding salt-and-pepper noises to the input images [28]. In the database literature, HAL [34] creates a similar ad-hoc strategy for training ML models for index-tuning. AUG adds a Gaussian noise (with a standard deviation of 10% of each column's value range) to the value in each clause, and it requires computing ground truth labels for the synthetic queries.

Hard Example Mining (HEM) strengthens the ML model where it fails and has been widely used in domains such as image object detection [12, 42]. In our experiments, HEM evaluates a previously trained model on the newly arrived queries and updates the model using the queries weighted by evaluation error. We apply the random noise described in AUG to robustly build HEM. As a result, HEM requires computing ground truth labels for the new queries.

Mixture (MIX) is another format of FT to augment the newly arrived examples. It combines queries from the initial training and the new workloads to update the CE model, which improves the generalizability of the model. Without generating synthetic samples and computing additional labels, MIX can be helpful based on the similarity between the training and the testing distributions.

CE models used in Warper**.** Recall that Warper is agnostic to the CE model and works for previously trained models as black-boxes. We show results with three ML-based estimators.

LM [10] models take a range predicate as input and predict its cardinality. The input $\texttt{q} = \{\texttt{low}_1, .., \texttt{low}_d, \texttt{high}_1, .., \texttt{high}_d\}$ represents a conjunction of range predicates on d columns (more details in §2). We use the min and max value of a column to represent one sided predicates. We follow the logic in [10] - for columns with categorical values, predicates are integer dictionary identifies, and we quantized each column into [0,1K] using equi-width bucketing. The model is about 64KB in size. Note that we do not use additional features produced by SQL Server as in [10], since they require a long time unless SQL Server is customized; we defer more details to the paper [10]. In our experiments, we use a few variants of LM including one with multi-layer perceptrons (MLPs) and one with gradient boost trees (GBTs), namely `LM-mlp` and `LM-gbt`. These two models have the same input and output; however, `LM-mlp` updates the model by fine-tuning while `LM-gbt` uses re-training. We implement both with sklearn. MLP updates with a batch size of 32 and a learning rate of $1e^{-3}$, while GBT uses a learning rate of $1e^{-2}$. More variants will be discussed in §4.1.2.

MSCN [25] learns a more complex model which uses query predicates, join conditions, and bitmaps as input. The model consists of a pooling layer on each input and an MLP, which produces cardinality estimates. For single-table CE, we use a simplified version here by removing the join condition and bitmap inputs. We use the same predicates and ground truth as LM. To examine join CE, we construct newly arrived queries by randomly sampling the join conditions and use the same procedure above to generate predicates on base tables. MSCN models are 64KB in size and update using fine-tuning. We use PyTorch for implementation with a batch size of 32 and a learning rate of $1e^{-3}$.

Evaluation method. We evaluate Warper and the baselines with different data and workload drifts `c1-3` in Table 2, and the same CE model is used in all methods. For `c4`, Warper falls back to FT (§3.1) and we do not evaluate explicitly. We run each experiment 10 times and report aggregated metrics, including error and adaptation efficiency. We evaluate each adaptation method at 0,20%,..,100% of our test period. n_t is then computed relative to time spent and

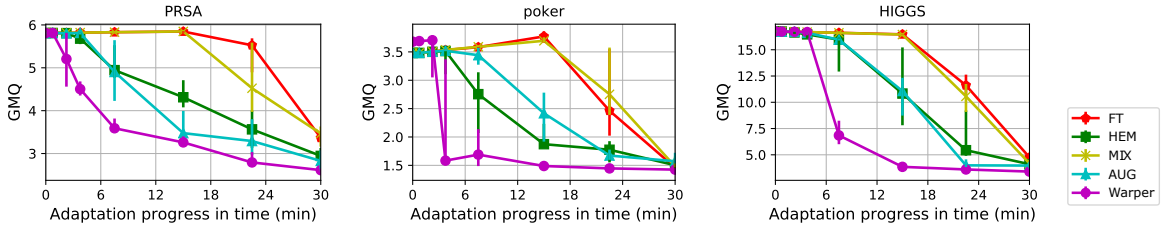

Figure 6: Comparison of handling workload drifts (c2) using LM-MLP. We show GMQ on the hold-out test set at different adaptation steps with one new query arriving every five seconds. Table 7a shows the relative speedups Δ. We plot the first and the third quarters on the error bar.

Dataset		PRSA			Poker			HIGGS		
Method		AUG	HEM	Warper	AUG	HEM	Warper	AUG	HEM	Warper
Annotation cost*		0.01s/query			0.03s/query			0.39s/query		
Model building cost*		-	1s	52.1s	-	1s	60.5s	-	1s	58.5s
Avg CPU Usage	10 min @ 10 q/s arrival	0.27%	0.27%	1.0%	0.74%	0.75%	1.58%	9.95%	9.96%	10.77%
	10 min @ 1 q/s arrival	0.03%	0.03%	0.75%	0.07%	0.07%	0.90%	0.95%	0.96%	1.77%
	30 min @ 0.2 q/s arrival	0.005%	0.01%	0.25%	0.015%	0.019%	0.29%	0.20%	0.20%	0.47%

Table 6: We show cost overhead to adapt a CE model. *: Costs in a single thread. Given different arrival rates of new queries, FT requires a minimum of 10-30 mins to fully adapt. Using the same amount of time and newly arrived queries and small CPU in extra, Warper **achieves better accuracy than the baselines.**

Exp.	Dataset	Cs	Wkld	Model	δ_m	δ_{js}	$\Delta_{.5}$	$\Delta_{.8}$	Δ_1
a.Wkld drift (Figure 6)	PRSA	c2	w12/345	LM-mlp	2.4	0.31	7.4	4.8	3.1
	Poker	c2	w12/345	LM-mlp	2.0	0.27	7.1	7.3	7.7
	Higgs	c2	w12/345	LM-mlp	12	0.60	3.8	3.7	3.5
b.Different models	PRSA	c2	w12/345	LM-gbt	0.8	0.31	1.1	1.0	1.0
	Poker	c2	w12/345	LM-gbt	1.3	0.27	1.0	3.5	6.8
	Higgs	c2	w12/345	LM-gbt	9.9	0.6	1.0	1.0	1.2
	PRSA	c2	w12/345	LM-ply	0.5	0.3	1.9	1.1	1.1
	Poker	c2	w12/345	LM-ply	1.7	0.3	1.0	1.0	1.1
	Higgs	c2	w12/345	LM-ply	6.1	0.6	1.0	4.0	1.5
	PRSA	c2	w12/345	LM-rbf	2.6	0.3	1.5	1.5	1.3
	Poker	c2	w12/345	LM-rbf	1.6	0.3	2.2	4.3	5.8
	Higgs	c2	w12/345	LM-rbf	11.5	0.6	1.2	1.3	1.2
	PRSA	c2	w12/345	MSCN	1.8	0.31	6.2	3.6	3.9
	Poker	c2	w12/345	MSCN	1.4	0.27	6.0	8.1	3.3
	Higgs	c2	w12/345	MSCN	9.6	0.6	2.5	5.2	3.2
c.Different drifts	PRSA	c1	w1-5	LM-mlp	0.4	0	3.0	7.6	1.0
	Poker	c1	w1-5	LM-mlp	0.9	0	1.3	1.1	1.5
	Higgs	c1	w1-5	LM-mlp	12	0	1.5	1.0	1.0
	PRSA	c3	w12/345	LM-mlp	2.1	0.31	1.1	1.1	1.0
	Poker	c3	w12/345	LM-mlp	1.9	0.27	1.4	1.4	1.2
	Higgs	c3	w12/345	LM-mlp	0.5	0.60	1.0	1.0	1.0
d.Join CE	IMDB	c2	w4/w1	MSCN	72	0.52	2.1	2.8	1.1

Table 7: Warper **with different CE models, drifts and workload distributions; results are aggregated over 10 runs.**

query arrival rate. Warper, AUG and HEM synthesize $n_g = 10\% n_t$ queries in each adaptation step to lower the annotation cost; we will show a sensitivity analysis in §4.3 with different choices of n_g and n_t. Warper uses a fixed $n_p = 1K$ in the picker; AUG and HEM randomly sample the same number of queries from different distributions to match Warper.

4.1.1 Results and discussions. We first examine workload drift c2 in which all Warper components are used, whereas other cases use only a subset (see §3). Evaluations of other CE models, types of drift and workload changes will be discussed in §4.1.2.

Adaptation speedups. Table 7a demonstrates model adaptation on three datasets with LM-mlp, while Figure 6 shows the progress at different adaptation steps. Our observations are two-fold.

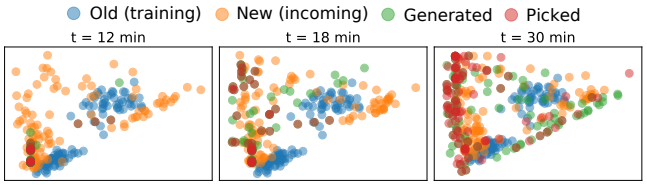

Figure 7: Visualization of adaptation on the PRSA dataset with c2 drift and workload w12/345.

First, we note that speedups here are already agnostic to and normalized by the query arrival rate. When more queries from the new workload arrive, all adaptation methods improve the accuracy and Warper adapts faster than other baselines. For example, on the PRSA dataset, Warper reduces the GMQ to 4.8 (i.e., 50% of possible improvement) at 3.3min; Warper provides a $\Delta_{0.5} = 7.4\times$ speedup at this accuracy compared with FT at 25min. We observe large speedups provided by Warper on all test datasets. With higher accuracy required, the gains are less in general; e.g., Δ_1 decreases to 3.1× on the PRSA dataset. The last bit of accuracy improvements may have to come from the real incoming queries.

Next, MIX occasionally outperforms FT slightly without additional queries used in the model update; HEM and AUG perform better with additional queries but are not as good as Warper. These adaptation solutions are ad-hoc and often require careful heuristics design to generate new queries. In practical systems, the use cases of these baselines remain unclear.

Qualitative results. Figure 7 demonstrates different sets of queries using the PCA visualization discussed in §2. As the adaptation proceeds over time, we find that the generated (in green) and picked queries (in red) in general follow the incoming query distribution (in orange). A small portion of generated queries near the middle of the diagonal do not follow either old or new distributions; we found that these queries help the adaptation.

Exp.	Wkld	$\delta_{\mathtt{m}}$	$\delta_{\mathtt{js}}$	$\Delta_{.5}$	$\Delta_{.8}$	Δ_1
d. Different workload	w1/2	1.1	0.35	3.8	3.8	4.0
	w1/3	16.0	0.43	3.2	4.4	6.2
	w1/4	5.2	0.41	5.2	4.8	6.2
	w2/3	13.7	0.32	7.9	5.5	4.2
	w2/4	5.2	0.34	8.9	8.6	8.0
	w5/3	14.7	0.30	4.7	5.7	3.4
	w5/4	4.4	0.22	4.6	3.8	1.6
	w34/125	0.1	0.28	1.3	1.5	1.5
	w35/124	0.2	0.26	4.5	1.2	1.1
	w125/34	9.2	0.25	12.7	12.1	1.4

Table 8: Warper **with different workload distributions on PRSA.**

Adaptation costs. Here, we measure the costs incurred by Warper. Table 6 breaks down the costs of Warper and alternatives at various query arrival rates. Note that all these methods update the CE model, which only takes a few seconds. FT and MIX do not incur any additional cost unlike the other methods.

At each adaptation step, Warper's costs consist of updates to internal components, generating synthetic queries when needed ($<$ 1 second), and computing the ground truth. To adapt a CE model to the workload drift, Warper incurs a compute overhead of about 0.25% to 10.8% CPU usage depending on the query arrival rate. For any arrival rate, a system can choose to (1) adapt immediately, which has the highest instantaneous CPU usage, or (2) spread the adaptation cost over a longer duration. For instance, when 10 queries arrive per sec on the PRSA dataset, Warper may use 1.8% CPU over 10 min, or use 0.47% over 30 mins. Furthermore, these numbers are from an unoptimized prototype in python; cost reduction optimizations could be helpful in future work. Nevertheless, the costs are already small and insignificant in comparison to the alternatives and do not hold back Warper from keeping up with all the cases in Table 6.

Note that when new queries arrive at a higher rate (e.g., 1K q/s in 10 mins) than the cases shown in Table 6, Warper cannot keep up (CPU usage >100%). Warper either spreads the adaptation in a longer period or uses `c3` or `c4` mode with `det_drft` (§3).

AUG and HEM annotate additional queries and are cheaper than Warper but do not adapt as fast as Warper. FT and MIX do not use additional queries and are the most efficient solutions; however, they do not offer fast adaptation as Warper and other baselines.

4.1.2 Generalization in Warper. Beyond the `c2` case shown above, we are interested in how Warper can generalize to other models, types of drifts and workload changes.

Adapting for different models. Instead of `LM-mlp`, we now use `LM-gbt`, with a Gradient Boosting Tree regressor which re-trains, and `MSCN` which fine-tunes for single-table CE. In addition, we use `LM-ply` with a 5-degree polynomial-kernel SVM as the regression model in LM, as well as `LM-rbf` with a Radial Basis Function (RBF)-kernel SVM [6]. Other settings remain unchanged. Table 7b shows the results on two datasets and we show Δ speedups at different accuracy targets. Relative to fine-tuning or re-training using the same CE model and dataset, Warper still provides promising speedups for `MSCN`. On the PRSA and Higgs datasets, Warper only slightly improves adaptation for `LM-GBT`. In all cases, Warper performs no worse than FT or RT. Warper can help and accelerate adaptation for these different models. We also found that a linear-kernel SVM did not work as a CE model (has a high error) and hence does not worth further adaptation; this is as expected since predicates are non-linear [10].

Adapting to different drifts. In a data drift `c1`, all labels from the training set $\mathbb{I}_{\mathtt{train}}$ become outdated. For the data drift experiment, we sort the dataset by one column and truncate the table in half to differentiate the data distributions. The ground truth labels are computed by querying the updated data table. The test workload remains unchanged as $\mathbb{I}_{\mathtt{train}}$; Warper picks useful queries to label (recall the picker described in §3.2) and updates the CE model. Our experiments similarly run periodically; we compare Warper with FT that fine-tunes $\mathbb{M}$ by randomly picking the same numbers of queries to annotate from the training set. Table 7b demonstrates the results. At different accuracy targets, we observe various speedups due to saved annotations.

In a workload drift `c3`, ground truth labels from the training set $\mathbb{I}_{\mathtt{train}}$ are valid, but the newly arrived queries are not companied with `gt` labels. The Warper picker works similarly as in `c1`. Our experiment runs periodically to pick and annotate queries from the newly arrived queries. At each adaptation step, we compare Warper against FT, which uniformly picks the same amount of queries at random for annotation. Table 7c demonstrates the results and Warper adapts faster with a fixed annotation budget compared with FT on randomly annotated samples.

In both cases above, Warper only uses the picker; the overhead caused by Warper is only updating the learned modules and is smaller than `c2` as shown earlier and in the extended report [2].

Adapting for different workload changes. We consider the case of `c2` but vary the training and new workloads as illustrated in Table 8. As shown in the results, different workload distributions exhibit different adaptation speedups with median $\Delta_{0.5}, \Delta_{0.8}, \Delta_1$ being 4.7, 4.6 and 3.7 while recall the results in §4.1.1 for 7.4, 4.8 and 3.1 respectively. Notably, speedups are less significant when the accuracy gap is already small (e.g., $\delta_{\mathtt{m}} \leq 0.2$). We also observe that the accuracy gap in drifts $\delta_{\mathtt{m}}$ can be uncorrelated with the intrinsic distribution difference $\delta_{\mathtt{js}}$, but both are useful metrics. CE models that are explainable remain an open question among learned database components. Figure 8 further demonstrates the adaptation progress on various datasets and query distributions. To this end, we consider Warper useful and robust to different distributions of workload drift.

Adapting for join CE. As shown in Table 7d, we examined join cardinality estimation with the MSCN model on the IMDB dataset with an arrival rate of one new query per minute; other settings remain unchanged as above. We randomly generated 16K join queries to pre-train the MSCN estimator with a 128K storage budget. Warper achieved $\Delta_{0.5}, \Delta_{0.8}$ and $\Delta_{1.0}$ at 2.1x, 2.8x, 1.1x. That said, we consider Warper generic and agnostic to various CE models that need to be adapted.

Remark. Warper offers a reasonable cost-speed tradeoff and can be a useful alternative to FT when the cost overhead is affordable. We report supplementary results in the extended report [2]; we observe similar findings on other datasets and tasks.

In all test cases and datasets (Table 7 and 8), we observe that Warper performs no worse than FT ($\Delta \geq 1$), and applying Warper is

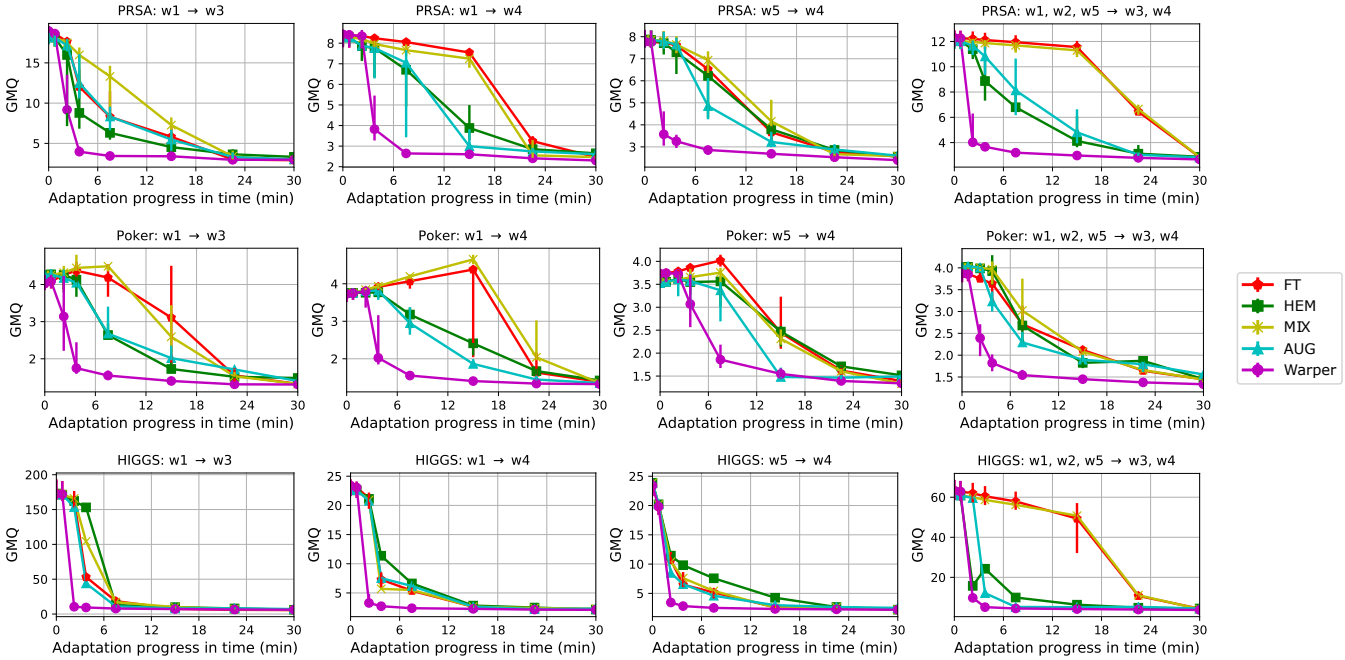

Figure 8: We demonstrate adaptation in workload drift `c2` using `LM-MLP` with training and newly arrived workloads described in the figure title, e.g., `w1` → `w3` indicates that the model is trained on workload `w1` and is tested when the workload drifts to `w3`.

less likely to cause degradation. This is perhaps because the newly arrived queries are still used to update the CE model as in FT.

4.2 Can Warper deliver end-to-end gains?

Here, we consider how model adaptation affects end-to-end query performance. Query plans of complex queries can be hard to analyze; hence, for ease of description, we show results on queries for the simple select-project-join template shown in Figure 1 over the `Lineitem` and `Orders` tables of TPC-H at a scale factor of 10. Our results use a production query optimizer (QO); as shown in Table 9, we perform three different experiments which aim to trigger the following plan changes:

`S1` *Buffer spills.* The intermediate results of a join input will spill to a temporary table when the predicate cardinality is underestimated; spills are wasteful and delay the query execution. Over-estimates waste memory but have little impact on latency.

`S2` *Nested loop vs. hash join.* When both join inputs are estimated to have a small cardinality, the QO picks nested loop joins over hash joins. Underestimates significantly degrade latency since nested loop join is inefficient when the inner loop is large. Here too, overestimates have only a minor latency impact.

`S3` *Choice of which side to build the bitmap.* In parallel (multi-threaded) executions, the QO builds a bitmap on the join input with the smaller estimated cardinality and applies the bitmap on the other input to reduce the number of rows that go through the join operation. Choosing the wrong input on which to build the bitmap can degrade latency.

We generate 100 test queries from the same template that is used in training and run them using single or multiple threads as described in Table 9. We do not use indexes here since adding indexes further complicates the query plan choice. Table 9 also shows that `S1-3` have different severity of latency regression; for example, with `S1`, the maximum latency degradation is 2.1×, i.e., queries took over twice as long to finish when using imperfect cardinality estimates compared to the plans generated when using the true cardinalities.

Query setting	Executed as:	Predicate on	Latency gap
`S1` - Buffer spill	Single thread	L	2.1×
`S2` - Join type	Single thread	L, O	306×
`S3` - Bitmap distr.	Multi-thread	L, O	5.3×

Table 9: Queries used in this section. Latency gap indicates the max latency difference between plans with accurate and inaccurate CE.

We build a CE model for predicates on both tables and study how model adaptation affects CE errors and thus end-to-end query latency in continuous drifts of three kinds. The seed CE model that we begin with has been trained with workload `w1` from Table 8. Each drift demonstrates a case from §3.1 and Figure 2: (1) Drift A changes workload from `w1` to `w2`; see Table 8; (2) Drift B is also a workload drift that changes just the first half of each period to `w4`, and (3) Drift C combines a workload drift (back to `w1`) with a data drift (as described in §4.1.2).

Our experiment ran periodically as in the previous section, and we measured the GMQ of the test queries in each adaptation step. We also measure the average query latency by executing the query plans that would have been generated using the predicted cardinalities (we do so by changing the cost estimates of the memo groups accordingly inside the QO). We clear all system caches between successive query executions.

Figure 9 shows the CE accuracy on predicates and the query latency for each of `S1-3`. Comparing Warper with FT which adapt at the same period, we draw the following conclusions.

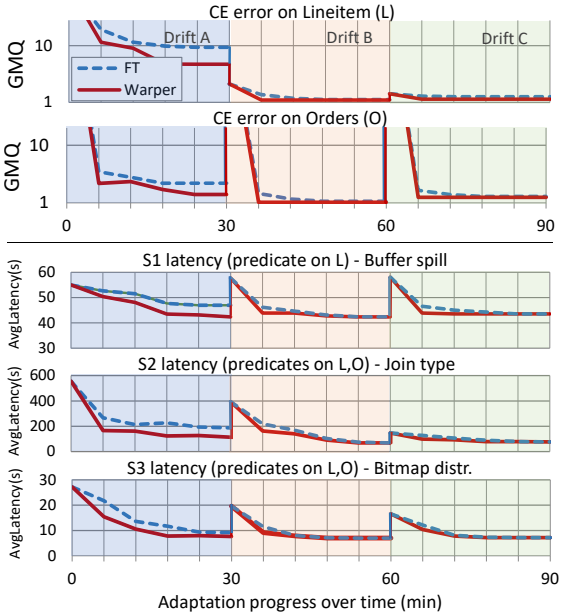

Figure 9: We show that faster model adaptation results in query performance gains in three cases and in continuous drifts.

First, drifts lead to higher GMQ and latency regressions due to suboptimal plan choice. We see that without adaptation the CE accuracy can be up to 1000× off. For the considered scenarios, `S1-3`, the average latency regression is 30%-300% without adaptation. When predicates are present on both join input and/or the chosen plan has a nested loop join, we observe up to 306× worse latency, which is perhaps catastrophic. As the figure shows, fast model adaptation greatly reduces latency regressions by offering better cardinality estimates sooner.

Next, we observe that different drifts have different effects. Across `S1-3` and across all drifts, Warper adapts faster than FT. The speedup is particularly significant in `S1` Drift A, where Warper converges in about 12 minutes while FT does not catch up even at the end of 30 minutes. Warper reduces by more than half the total duration for which queries regress. The improvements in other scenarios are sizable as well.

We note that it is non-trivial to connect improvements in the accuracy of cardinality estimates to improvements in query latency, especially for complex queries [4]. Our results show a few simple cases where the gains are sizable, but it is not clear what the improvements will be with a different query optimizer, a different DBMS engine, or a different query set. We also point out that the drifts used here are hypothetical, and it is not fully clear what types of drifts occur in practice. Nevertheless, we point out that an inexpensive and fast adaptation would be a useful addition, although more evaluation is necessary to quantify the added value.

4.3 Ablation and hyperparameter analyses

Similar as §4.1.1, we leverage workload drift `c2` in which all Warper components are used. This facilitates the ablation and sensitivity analyses in this section.

	Dataset	Warper	$\mathbb{P}\rightarrow$rnd pick	$\mathbb{P}\rightarrow$entropy	$\mathbb{G}\rightarrow$AUG
$\Delta_{0.8}$	PRSA	4.8	3.3	3.8	4.6
	Poker	7.3	1.3	6.7	6.9
Δ_1	PRSA	3.1	2.0	3.2	3.2
	Poker	7.7	1.0	1.6	6.9

Table 10: Replacing learned Warper components with alternatives.

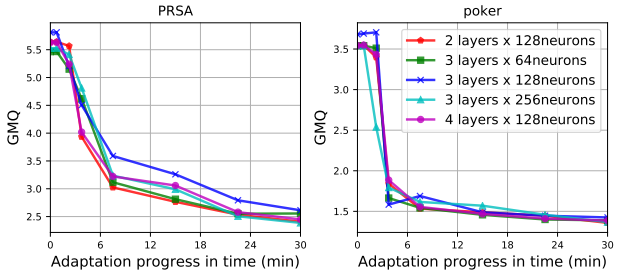

Figure 10: Varying the NN hyperparameters in $\mathbb{E}$ and $\mathbb{G}$.

Understanding Warper components. We perform an ablation study here and Table 10 evaluates two variants of Warper. For workload drift `c2`, we replace the query generator $\mathbb{G}$ and the picker $\mathbb{P}$ with alternatives. That is, we replace $\mathbb{G}$ with adding Gaussian noise to the newly arrived queries and replace $\mathbb{P}$ with simply picking the queries uniformly at random. Results on two datasets show that both variants demonstrate slower adaptation than Warper. Hence, we consider the generator and picker in Warper to be necessary.

We also replace the picker in Warper with an entropy-based active learning model which performs uncertainty sampling by calculating and weighting the cross-entropy for each query; queries with higher entropy have higher probabilities to be selected. The entropy-based method performs better than the naive random sampling picker but is inferior to the picker used in Warper.

Model hyperparameters. We show different NN structures in Figure 10 to replace the structures used in Table 3. Results indicate that hyperparameter tuning may improve the performance but concrete choices are unclear at this point; we aim at a simple and effective solution in this paper and we defer hyperparameter tuning to future work.

Cost analysis and budgeting in Warper. Let $\mathcal{B}$ be the cost budget on a CPU and $c_{\texttt{gt}}$ be the cost to annotate additional queries. Costs in a Warper adaptation step can be summarized as: $c_{\texttt{gen}} + c_{\texttt{pick}} + c_{\texttt{gt}} + c_{\texttt{AE}} + c_{\texttt{GAN}} + c_{\texttt{Model}} \leq \mathcal{B}$. In practice, $c_{\texttt{gen}}$ and $c_{\texttt{pick}}$, which depict the costs to generate and pick queries respectively, are both small and can be done within a second since we use simple models. $c_{\texttt{Model}}$, the cost to update the CE model, is a constant overhead no matter if Warper kicks in. $c_{\texttt{AE}}$ is a constant overhead to compute $\mathcal{L}_{\texttt{AE}}$ and to update related models, which depends on the model sizes and other hyperparameters in the auto-encoder training. $c_{\texttt{GAN}}$ is also a constant overhead to compute $\mathcal{L}_{\texttt{GAN}}$ and to update related models, which depends on the model sizes and the number of queries generated in the GAN update loop. We use

$$c_{\texttt{gt}} + C \leq \mathcal{B}$$

as a proxy to the cost, while C can be measured by runtime profiling, and $c_{\texttt{gt}}$ is nearly linear to the number of queries that need to be labeled n_a. From this formulation, we can see that with different

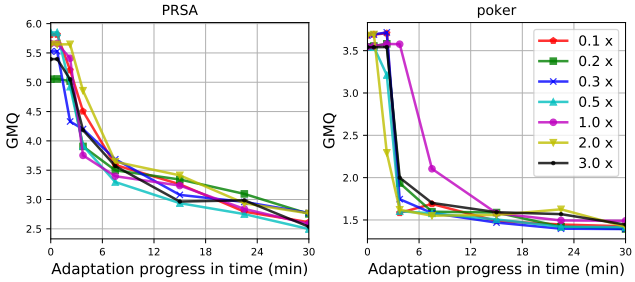

Figure 11: Trading compute for adaptation speedup. We vary the number of generated queries n_g and show the speedups. $n\times$ indicates $n_g = n \times n_t$ generated.

Dataset	PRSA				Poker			
n_g	0.1×	0.3×	1×	3×	0.1×	0.3×	1×	3×
Anno.	1.2s	3.6s	12.1s	36.3s	3.2s	9.6s	31.9s	95.7s
Model	const 52.2s (0.24% CPU usage)				const 60.6s (0.28% CPU usage)			
CPU	0.25%	0.26%	0.30%	0.41%	0.29%	0.33%	0.43%	0.72%

Table 11: Trading compute for adaptation speedup. We show the additional CPU utilization when n_g varies as multiples of n_t with an adaptation period 30min and one query arriving per five seconds.

query arrival rates, if the CPU usage is within the budget and Warper keeps up in compute, the relative adaptation speedups remain the same; the only difference is in the CPU usage ratio. Warper uses more CPU with a higher query arrival rate – more is spent in computing ground truth labels.

A key question now is how many queries n_g should the generator produce to update $\mathbb{M}$, because only these queries will affect the Warper cost due to annotation. In Figure 11 and Table 11, we test the effect of using different numbers of generated queries relative to the number of observed queries n_t. Results with workload drift `c2` at different adaptation steps show that using more generated queries does not necessarily accelerate the model adaptation but will increase the CPU utilization due to more queries being generated and annotated. In practice we use $n_g = 10\% n_t$ in each minibatch for a low annotation cost[12]. As shown in Table 6, for larger datasets, the annotation may become a large overhead when generating more queries. For example, on the HIGGs dataset, when generating $n_g = n_t$ queries, the overall CPU utilization goes up from 0.47% to 2.3% over 30 mins when one query arrives every five seconds. While improving the annotation efficiency for each query predicate (e.g., by using AQP or the technique in [9]) is orthogonal to this paper, a constant cost C still needs to be paid; when the budget $\mathcal{B}$ is less than C, which translates to overall 0.25% CPU utilization in our experiments over 30 mins, we recommend using FT/MIX that minimizes overhead.

5 RELATED WORKS

Learned CE models. Recent advances have exploited various machine learning models to provide fast and accurate cardinality estimations [10, 17, 44, 48]. In data and workload drifts, prior solutions have to re-train the model at different costs using the observed new queries. We focus on a generic solution that treats the CE models as black boxes with only input queries. Warper shows a faster adaptation compared with re-training or fine-tuning existing models, as well as augmentation using different strategies.

Handling drifts in active learning. Active learning [41] has been widely applied in production such as ads and recommendation systems [32], especially in a streaming setting; [16] provides a survey on different aspects of active learning with good depth and width. A machine learning model learns from a distribution $p(y|X)$ in which X is the training example and y is the ground truth label. A change in the distribution of X is often referred as *concept drift*, and a change in $p(y|X)$ as *covariant drift*. For instance, [26] provides several examples of data drifts on image and molecular datasets. In the CE problem, the data table D is an additional hidden variable, i.e., a learned CE model captures $X, D \rightarrow y$ that is conditioned on both workload and data table. That said, changing data X or changing workload D are both concept drifts. Different solutions have been proposed to (1) build ensemble models and perform *model selection* so that the best model is used for the current input pattern, or (2) use an *augmented workload* that combines observation windows from the training and testing streams and hence updates the ML model periodically. Recent augmentation solutions [34, 38, 47] also generate new examples of X, compute $p(y|X)$ and update the ML model, which can bring unseen generated queries and their annotations to help.

GANs. One related use case is in generating synthetic data to reduce annotation costs [5, 45]. Unlike prior solutions in which annotation costs are often sparse or delayed, generating new examples incurs immediate annotation costs for learned database components. On the other hand, GANs have been applied in other problems in DMBS. [11] leverages GAN to synthesize relational data that has a similar distribution as the original data table while preserving privacy during model training and generation. To the best of our knowledge, Warper is the first system for mitigating data and workload drifts for learned database components.

6 CONCLUSIONS

We demonstrate a machine learning system Warper that aims to improve a previously trained cardinality estimation model in the context of data and workload drifts. The key ideas include using generated queries from a Generative Adversarial Network (GAN) and picking among the available queries to annotate. We show that with small computational costs, Warper has good applicability and accelerates model adaptation which translates to query performance gains. Practitioners should consider Warper as an option to improve existing CE models if the cost overhead is affordable.

REFERENCES

[1] 2022. TPC-H Benchmark. http://www.tpc.org/tpch/.
[2] 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts - Extended Report. http://www.beibinli.com/docs/warper_extended_report.pdf
[3] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2001. STHoles: A Multidimensional Workload-aware Histogram. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 211–222.
[4] Surajit Chaudhuri. 1998. An Overview of Query Optimization in Relational Systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 34–43.
[5] Jaehoon Choi, Taekyung Kim, and Changick Kim. 2019. Self-ensembling with GAN-based Data Augmentation for Domain Adaptation in Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern*

[12] Warper disables the generator when $n_g < 1$ in our experiments.

Recognition. 6830–6840.

[6] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector Networks. *Machine learning* 20, 3 (1995), 273–297.

[7] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. 2015. Learning in Nonstationary Environments: A Survey. *IEEE Computational Intelligence Magazine* 10, 4 (2015), 12–25.

[8] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml.

[9] Anshuman Dutt, Chi Wang, Vivek Narasayya, and Surajit Chaudhuri. 2020. Efficiently Approximating Selectivity Functions Using Low Overhead Regression Models. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2215–2228.

[10] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates Using Lightweight Models. *Proceedings of the VLDB Endowment* 12, 9 (2019), 1044–1057.

[11] Ju Fan, Junyou Chen, Tongyu Liu, Yuwei Shen, Guoliang Li, and Xiaoyong Du. 2020. Relational Data Synthesis Using Generative Adversarial Networks: A Design Space Exploration. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1962–1975.

[12] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. 2008. A Discriminatively Trained, Multiscale, Deformable Part Model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.

[13] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. 2018. GAN-based Synthetic Medical Image Augmentation for Increased CNN Performance in Liver Lesion Classification. *Neurocomputing* 321 (2018), 321–331.

[14] Jerome H Friedman. 2002. Stochastic Gradient Boosting. *Computational statistics & data analysis* 38, 4 (2002), 367–378.

[15] Sylvia Frühwirth-Schnatter. 1994. Data Augmentation and Dynamic Linear Models. *Journal of Time Series Analysis* 15, 2 (1994), 183–202.

[16] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 1–37.

[17] Lise Getoor, Benjamin Taskar, and Daphne Koller. 2001. Selectivity Estimation Using Probabilistic Models. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT press.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*. 2672–2680.

[20] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Long text generation via adversarial training with leaked information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[21] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *Proceedings of the VLDB Endowment* 13, 7 (2020).

[22] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. 2018. Cycada: Cycle-Consistent Adversarial Domain Adaptation. In *International Conference on Machine Learning*. PMLR, 1989–1998.

[23] Weixiang Hong, Zhenzhen Wang, Ming Yang, and Junsong Yuan. 2018. Conditional generative adversarial network for structured domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1335–1344.

[24] Tero Karras, Samuli Laine, and Timo Aila. 2019. A Style-based Generator Architecture for Generative Adversarial Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4401–4410.

[25] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. *Proceedings of the 2018 Conference on Innovative Data Systems Research (CIDR)* (2018).

[26] 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.

[27] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 489–504.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 25. 1097–1105.

[29] Solomon Kullback. 1997. *Information Theory and Statistics*. Courier Corporation.

[30] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brébisson, Yoshua Bengio, and Aaron C Courville. 2019. Melgan: Generative adversarial networks for conditional waveform synthesis. In *Advances in Neural Information Processing Systems*. 14910–14921.

[31] 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.

[32] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. 2015. Recommender system application developments: a survey. *Decision Support Systems* 74 (2015), 12–32.

[33] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1493–1508.

[34] Lin Ma, Bailu Ding, Sudipto Das, and Adith Swaminathan. 2020. Active Learning for ML Enhanced Database Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 175–191.

[35] Christopher Manning and Hinrich Schutze. 1999. *Foundations of Statistical Natural Language Processing*. MIT press.

[36] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul23. 2019. Neo: A Learned Query Optimizer. *Proceedings of the VLDB Endowment* 12, 11 (2019).

[37] Magnus Müller, Guido Moerkotte, and Oliver Kolb. 2018. Improved Selectivity Estimation by Combining Knowledge from Sampling and Synopses. *Proceedings of the VLDB Endowment* 11, 9 (2018), 1016–1028.

[38] Hieu T Nguyen and Arnold Smeulders. 2004. Active Learning Using Pre-clustering. In *Proceedings of the twenty-first International Conference on Machine Learning*. 79.

[39] Alexander J Ratner, Stephen H Bach, Henry R Ehrenberg, and Chris Ré. 2017. Snorkel: Fast Training Set Generation for Information Extraction. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1683–1686.

[40] Veit Sandfort, Ke Yan, Perry J Pickhardt, and Ronald M Summers. 2019. Data Augmentation Using Generative Adversarial Networks (CycleGAN) to Improve Generalizability in CT Segmentation Tasks. *Scientific reports* 9, 1 (2019), 1–9.

[41] Burr Settles. 2009. Active Learning Literature Survey. (2009).

[42] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. 2016. Training Region-based Object Detectors with Online Hard Example Mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 761–769.

[43] PY Simard, D Steinkraus, and JC Platt. 2003. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis.. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings*. IEEE, 958–963.

[44] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. 2013. Efficiently Adapting Graphical Models for Selectivity Estimation. *The VLDB Journal* 22, 1 (2013).

[45] Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. 2018. Low-shot Learning from Imaginary Data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7278–7286.

[46] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal Component Analysis. *Chemometrics and Intelligent Laboratory Systems* 2, 1-3 (1987), 37–52.

[47] Donghui Yan, Ling Huang, and Michael I Jordan. 2009. Fast Approximate Spectral Clustering. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 907–916.

[48] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *Proceedings of the VLDB Endowment* 13, 3 (2019), 279–292.

[49] Dong Yu and Li Deng. 2016. *Automatic Speech Recognition*. Springer.

[50] Xiaojin Jerry Zhu. 2005. Semi-supervised Learning Literature Survey. (2005).