Accepted as a conference paper for SIGMOD 2023 @Seattle, WA, USA

Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data

Meghdad Kurmanji , Peter Triantafillou
University of Warwick, Coventry, UK
{meghdad.kurmanji,p.triantafillou}@warwick.ac.uk

ABSTRACT

Machine Learning (ML) is changing DBs as many DB components are being replaced by ML models. One open problem in this setting is how to update such ML models in the presence of data updates. We start this investigation focusing on data insertions (dominating updates in analytical DBs). We study how to update neural network (NN) models when new data follows a different distribution (a.k.a. it is "out-of-distribution" – OOD), rendering previously-trained NNs inaccurate. A requirement in our problem setting is that learned DB components should ensure high accuracy for tasks on old and new data (e.g., for approximate query processing (AQP), cardinality estimation (CE), synthetic data generation (DG), etc.).

This paper proposes a novel updatability framework (DDUp). DDUp can provide updatability for different learned DB system components, even based on different NNs, without the high costs to retrain the NNs from scratch. DDUp entails two components: First, a novel, efficient, and principled statistical-testing approach to detect OOD data. Second, a novel model updating approach, grounded on the principles of transfer learning with knowledge distillation, to update learned models efficiently, while still ensuring high accuracy. We develop and showcase DDUp's applicability for three different learned DB components, AQP, CE, and DG, each employing a different type of NN. Detailed experimental evaluation using real and benchmark datasets for AQP, CE, and DG detail DDUp's performance advantages.

KEYWORDS

Learned DBs, Out of Distribution Data, Knowledge Distillation, Transfer Learning

1 INTRODUCTION

Database systems (DBs) are largely embracing ML. With data volumes reaching unprecedented levels, ML can provide highly-accurate methods to perform central data management tasks more efficiently. Applications abound: AQP engines are leveraging ML to answer queries much faster and more accurately than traditional DBs [21, 42, 43, 65]. Cardinality/selectivity estimation, has improved considerably leveraging ML [17, 70, 77, 78, 84]. Likewise for query optimization [27, 44, 45], indexes [9, 10, 30, 49], cost estimation [63, 83], workload forecasting [85], DB tuning [34, 68, 81], synthetic data generation [7, 54, 76], etc.

1.1 Challenges

As research in learned DB systems matures, two key pitfalls are emerging. First, if the "context" (such as the data, the DB system, and/or the workload) changes, previously trained models are no longer accurate. Second, training accurate ML models is costly. Hence, retraining from scratch when the context changes should be avoided whenever possible. Emerging ML paradigms, such as active learning, transfer learning, meta-learning, and zero/few-shot learning are a good fit for such context changes and have been the focus of recent related works [20, 41, 74], where the primary focus is to glean what is learned from existing ML models (trained for different learning tasks and/or DBs and/or workloads), and adapt them for new tasks and/or DBs, and/or workloads, while avoiding the need to retrain models from scratch.

OOD Data insertions. In analytical DBs data updates primarily take the form of new data insertions. New data may be OOD (representing new knowledge – distributional shifts), rendering previously-built ML models obsolete/inaccurate. Or, new data may not be OOD. In the former case, the model must be updated and it must be decided how the new data could be efficiently reflected in the model to continue ensuring accuracy. In the latter case, it is desirable to avoid updating the model, as that would waste time/resources. Therefore, it is also crucial to check (efficiently) whether the new data render the previously built model inaccurate. However, related research has not yet tackled this problem setting, whereby *models for the same learning tasks (e.g., AQP, DG, CE, etc.) trained on old data, continue to provide high accuracy for the new data state* (on old and new data, as queries now may access both old data and new data, old data, or simply the new data). Related work for learned DB systems have a limited (or sometimes completely lack the) capability of handling such data insertions (as is independently verified in [70] and will be shown in this paper as well).

Sources of Difficulty and Baselines. In the presence of OOD, a simple solution is adopted by some of the learned DB components like Naru [78], NeuroCard [77], DBest++ [42], and even the aforementioned transfer/few-shot learning methods [20, 74]. That is to "fine-tune" the original model M on the new data. Alas, this is problematic. For instance, while a DBest++ model on the "Forest" dataset has a 95th percentile q-error of 2, updating it with an OOD sample using fine-tuning increases the 95th q-error to 63. A similar accuracy drop occurs for other key models as well – [70] showcases this for learned CE works. This drastic drop of accuracy is due to the fundamental problem of *catastrophic forgetting* [46], where retraining a previously learned model on new tasks, i.e. new data, causes the model to lose the knowledge it had acquired about old data. To avoid *catastrophic forgetting*, Naru and DBest++ suggest using a smaller learning rate while fine-tuning with the new data. This, however, causes another fundamental problem, namely *intransigence*, [6] whereby the model resists fitting to new data, rendering queries on new data inaccurate.

Another simple solution to avoid these problems would be to aggregate the old data and new data and retrain the model from scratch. However, as mentioned, this is undesirable in our environment. As a concrete example, training Naru/NeuroCard on the

"Forest" dataset (with only 600k rows) on a 40-core CPU takes ca. 1.5 hours. Similarly high retraining overheads are typically observed for neural network models, for various tasks. And, retraining time progressively increases as the DB size increases.

Therefore, more sophisticated approaches are needed, which can avoid *intransigence* and *catastrophic forgetting*, update models only when needed and do so while ensuring much smaller training overheads than retraining from scratch and at the same time ensure high accuracy for queries on old and new data. While for some tasks, like CE, some researchers question whether achieving very high accuracy through learned models will actually help the end-task (query optimization) [44], for tasks like AQP (which is itself the end-task) and for DG (with classification as the end-task) high accuracy is clearly needed, as shown here. Even for CE, with OOD data, accuracy can become horribly poor, as shown here, which is likely to affect query optimization.

1.2 Contributions

To the best of our knowledge, this work proposes the first updatability framework (DDUp) for learned DBs (in the face of new data insertions possibly carrying OOD data) that can ensure high accuracy for queries on new and/or old data. DDUp is also efficient and it can enjoy wide applicability, capable of being utilized for different NNs and/or different learning tasks (such as AQP, DG, CE, etc.). DDUp consists of a novel OOD detection and a novel model-update module. More specifically, the contributions of DDUp are:

- A general and principled two-sample test for OOD detection. Generality stems from it being based on the training loss function of the NNs. Compared to prior art, it introduces no extra costs and overheads, and could be used with different NNs, with different loss functions, in different applications. To further minimize detection time, it is divided into offline and online phases.
- A novel and general formulation of transfer-learning based on sequential self-distillation for model updating. This formulation allows a higher degree of freedom in balancing tasks w.r.t new and old data, can adapt to different models and tasks, and maximizes performance via self-distillation.
- Importantly, DDUp can be used by any pre-trained NN without introducing any assumptions on models or requiring additional components that might require to retrain models or incur more costs. Here, we instantiate it for three different tasks (namely, the CE task, using the Naru/NeuroCard deep autoregressive network (DARN) models [77, 78], the AQP task, using the DBEst++ mixture density network (MDN) model [42], and for the DG task, using the Tabular Variational AutoEncoder (TVAE) model [76]) each of which employs a different NN type. These are representative learning tasks and networks with evident importance in DBs and beyond. These instantiations are also novel, showing how to distil-and-update MDNs, DARNs, and TVAEs.
- Finally, DDUp is evaluated using six different datasets and the three instantiated learned DB components, for AQP, CE, and DG

1.3 Limitations

DDUp focuses only on data insertions, which are essential and dominant in analytical DBs, and not on updates in place and deletes, which are prevalent in transactional DBs. Nonetheless, the latter touch upon an open problem in the ML literature, namely "*unlearning*", where it typically concerns privacy (e.g., removing sensitive data from images in classification tasks) (e.g., [15, 61]). Studying unlearning for DB problem settings is a formidable task of its own and of high interest for future research.

Also, DDUp is designed for NN-based learned DB components. This is so as neural networks are a very rich family of models which have collectively received very large attention for learned DBs. Extending DDUp principles beyond NN models is also left for future research.

2 THE PROBLEM AND SOLUTION OVERVIEW

2.1 Problem Formulation

Consider a database relation R with attributes $\{A_1, A_2, ..., A_m\}$. This can be a raw table or the result of a join query. Also consider a sequence of N insertion updates denoted by $I = \{I_1, I_2, ...I_N\}$. Each I_t is an insert operation which appends a data batch $D_t = \{(A_1, A_2, ..., A_m)_t^{(i)}; i = 1, ..., n_t\}$ to R, where n_t is the number of rows. Let S_t be a sufficient sample of D_t and $S_{t-1}^{\leq}$ be a sufficient sample from $\cup_{j=0}^{t-1} D_j$. We naturally assume that $|R|$ is finite. And, due to the training restrictions of existing models, we also make the natural assumption:

$$\forall A_i \in R : supp(D_t(A_i)) \subseteq supp(D_{t-1}(A_i))$$

where $supp(D(A_i))$ is the support of attribute A_i in dataset D. This assumption satisfies the condition based on which the domain of each attribute is not violated in the upcoming update batches.

Statistical test for data changes. We define out-of-distribution detection as a two-sample hypothesis test between a sample of historical data and a sample of the new data. Let $S_{t-1}^{\leq}$ have a joint distribution of $P(A_1, \ldots, A1_m) \equiv \mathbb{P}$ and S_t have a joint distribution of $Q(A_1, \ldots, A_m) \equiv \mathbb{Q}$. We define the null hypothesis $H_0 : \mathbb{P} = \mathbb{Q}$ which asserts that S_t and $S_{t-1}^{\leq}$ are coming from a same distribution; and the alternative hypothesis $H_A : \mathbb{P} \neq \mathbb{Q}$ which declares that the two samples are generated by two different distributions.

Incrementally updating the model. Consider for I_0 a model M_0 is trained by minimizing a loss function $\mathscr{L}(D_0; \Theta_0)$. This model may be stale for $I_t; t > 0$. Ideally, the goal of incremental learning is: at time t train a model M_t that minimizes a function over $\sum_{i=1}^{t} \mathscr{L}(D_i; \Theta_i)$. This new model should not forget $\{I_i; i = 0, 1, ..., t-1\}$ and also learn I_t.

2.2 A High Level View of DDUp

The overall architecture of DDUp is depicted in Figure 1.

DDUp process batches of tuples at a time. Such batched handling of insertions is typical in analytical DBs. Furthermore, this takes into account that the effect of single tuples is usually negligible for the overall large space modelled by NNs. And, for most tasks like CE, AQP and DG, the effect of single tuples in the final result is very small, considering the large sizes of tables. And batching amortizes detect-and-update costs over many insertion operations.

Upon a new batch insertion, DDUp takes the latest model M_{t-1}, and performs a bootstrapping sampling from the previous data to

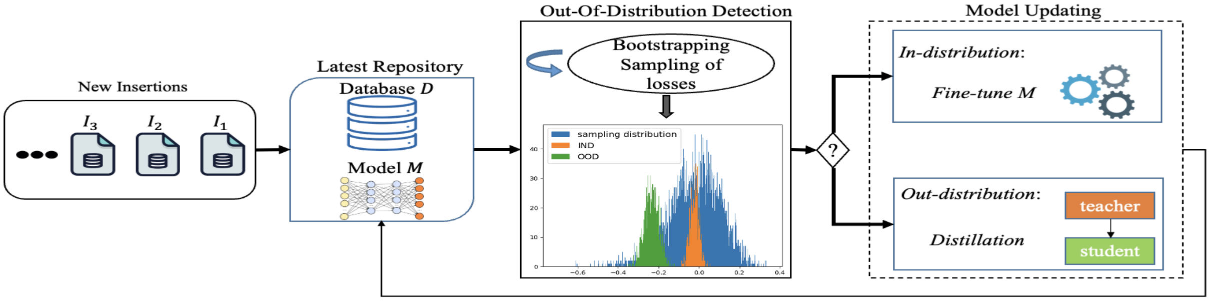


Figure 1: The overall structure of DDUp. DDUp uses the latest model and previous data to build a sampling distribution for the two-sample test, and updates the learned component based on the shift in the data distribution.

build the sampling distribution for the average loss values. DDUp uses this distribution to calculate a significance level corresponding to a confidence interval (e.g a 95th confidence interval). The general idea is that if the new data is similar to the previous data (IND in Figure 1), the loss values of M_{t-1} for this new data should lie within the threshold. This means that the new data has the same distribution and therefore the model could be left intact (updating maybe just the hyper-parameters of the system, including possible frequency tables and other table statistics. Alternatively, a simple fine-tuning can be performed to adapt the model to the new data.

If the loss values exceeded the threshold, this implies that the data distribution has significantly changed. DDUp will deploy a teacher-student transfer learning method based on knowledge distillation to learn this new distribution without forgetting the knowledge of the old data. In this framework, while the student directly learns the distribution of the new data, the teacher act as a regularizer to make the student also learn about the old distribution.

3 OUT-OF-DISTRIBUTION DETECTION

3.1 Background

In ML, OOD is typically addressed from a classification perspective. Formally, assume D is a dataset of (x, y) pairs which are drawn from a joint distribution, $p(x, y)$, where $x \in \mathcal{X} := \{x_1, x_2, \ldots, x_n\}$ is the input (independent variable) consisting of n features, and $y \in \mathcal{Y} := \{1, 2, \ldots, k\}$ is the label corresponding to one of the k in-distribution classes. A sample (x, y), that probably is generated by a different distribution than $p(x, y)$, is called OOD, if $y \notin \mathcal{Y}$, i.e it does not belong to any previously seen classes.

A similar problem has previously been addressed in statistics as *concept drift* detection, where different types of shifts are distinguished by expanding $p(x, y)$ using the Bayes rule:

$$p(x, y) = p(x)p(y|x) \tag{1}$$

Based on Eq. 1, changes in $P(y|x)$ are usually referred to as *Real drift*, while changes in $P(x)$ are called *virtual drift* [14]. In $X \rightarrow y$ problems the latter mostly is known as *covariate shift*. Deciding which drift to detect is dependent on the underlying models. For example, deep autoregressive networks (e.g., used by [78]) learn the full joint distribution of a table. Hence, they are sensitive to *covariate shift* upon insertions. On the other hand, mixture density networks (e.g., used by [42]), model the conditional probability between a set of independent attributes and a target attribute. Hence, for these models, one would be interested in detecting *real shift*.

3.2 Loss based OOD Detection

There are several challenges that make it difficult to simply adopt one of the OOD detection algorithms in the ML or statistical learning literature. First, DB tables are multivariate in nature and learned models are usually trained on multiple attributes. As a result, univariate two-sample tests like Kolmogorov–Smirnov (KS) test are not suitable for this purpose. Second, the test should introduce low overheads to the system as insertions may be frequent. Therefore, multivariate tests like kernel methods that require to learn densities and perform expensive inference computations are not desirable. Third, we aim to support different learning tasks for which different models might be used. Thus, most of OOD detection methods in ML that are based on probability scores (confidence) of classification tasks are not useful here. Moreover, the test should be able to adapt efficiently to the case where insertions occur within old data, that is, without having to recalculate baseline thresholds etc.

An efficient OOD detection method is now proposed that resolves all above issues by leveraging the underlying ML models themselves. Central to most learned data system components is the ability to derive from the underlying data tables a model for the joint or conditional data distribution like $p(x)$ or $p(y|x)$. A model usually achieves this by learning a set of parameters Θ that represent a function f by iteratively optimizing over a loss function as follows:

$$f_\Theta = \underset{f \in \mathcal{F}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x); \Theta) + \Omega(f) \tag{2}$$

where, Ω is a regularizer term, n is the number of samples, and f could be the outputs of the model in the last layer (called *logits*), or the probabilities assigned by a "softmax" function.

We will later discuss different loss functions in more details when instantiating different models. In general, loss functions are usually highly non-convex with many local mimina. However, a good learning strategy will find the global minimum. Because of the large data sizes, training is usually done by iterating over mini-batches and a gradient descent algorithm updates the parameters based on the average of loss of the samples in each mini-batch per iteration. For the rest of the paper, when we mention 'loss value' we mean average of losses of the samples in a batch. Once the model is trained, i.e. the loss values have converged, the model can serve as a transformer to map (high-dimensional) input data to the one-dimensional loss functions space around the global minimum.

Accordingly, the previous data (seen by the model) are closer to the global minimum compared to the out of distribution data.

The above discussion explains the possibility to compare in- and out-of distribution data just by relying on the underlying models without any further assumptions/components, in a low-dimensional space. With these in hand, we can perform a statistical testing to compare the loss values of old data and new data. In the following we will explain a two-sample test for this purpose.

3.3 A Two-Sample Test Procedure

The steps for a two-sample hypothesis test are: 1. Define the null, H_0, and alternative hypothesis, H_A. 2. Define a test statistic d that tests whether an observed value is extreme under H_0. 3. Determine a significance level $\delta \in [0, 1]$ that defines the *type*-1 *error* (false positives) of the test. 4. Calculate *p-value* which equals the probability that a statistical measure, e.g. distance between two distributions, will be greater than or equal to the probability of observed results. 5. If *p-value* $<= \delta$ then the *p-value* is statistically significant and shows strong evidence to reject H_0 in favor of H_A. Otherwise, the test failed to reject H_0.

The main challenge herein is how to calculate the test significance of the test statistic, i.e the *p-value*. As explained in Section 2, we aim to detect if the new data that is inserted to the system at time t has a different distribution than the previous data. Consider $S_{t-1}^{\leq}$ be a sample of the previous data and S_t be a sample of the newly inserted data. Let $d(S_{t-1}^{\leq},\ S_t)$ be a distance function that measures the distance between the two samples. Then, the test significance would be *p-value* $= P(d < d_t | H_0)$ where d_t is our test threshold.

Choosing the test statistic. The test statistic should reflect the similarity of new data to old data. According to our discussion in Section 3.2, we use the loss function values after the convergence of the models. We use a linear difference between the loss values of the two samples as our test statistics as follows:

$$d(S_{t-1}^{\leq}, S_t) = \frac{1}{|S_{t-1}|} \sum_{s \in S_{t-1}} \mathscr{L}(s; \Theta) - \frac{1}{|S_t|} \sum_{s \in S_t} \mathscr{L}(s; \Theta) \qquad (3)$$

where $\mathscr{L}$ is a loss function achieved by training model M with parameters Θ. From Eq. 3 follows that if the loss function is Negative Log Likelihood, and the likelihoods are exact, the test statistic will be the logarithm of the well-known *likelihood-ratio* test. Eq. 3 also gives intuition about the effect size: the larger d is, the larger the difference between two data distributions would be. Although many of the learned DB models are trained by maximizing likelihood, some other models (e.g., regressions) are trained using a *Mean-Squared-Error* objective. It has been shown [71] that MSE optimization maximizes likelihood at the same time. Therefore, the form of the distance function in Eq. 3 still holds. The important consequence of Eq. 3 is that, under i.i.d assumptions for both samples, it can be shown that the central limit theorem holds for distribution of d under the null hypothesis, hence, it has a normal limiting distribution with a mean at 0 and unknown standard deviation. To estimate the standard deviation (std), we utilize a bootstrapping approach.

3.4 Offline and Online Steps

The main performance bottleneck of such an OOD detection is bootstrapping. Fortunately, this part could be performed offline before data insertion. In the offline phase, we draw n bootstrap samples of size $|S_{t-1}^{\leq}|$ from $S_{t-1}^{\leq}$. (In practice, when we have access to the original data, we make n bootstrap samples of size $|S_{t-1}^{\leq}|$ from $D_{t-1}^{\leq}$). We use the model M_{t-1} to compute the average likelihoods (or other losses) of each sample and create a sampling distribution of said average likelihoods. Then, we calculate the standard deviation of the sampling distribution, std, which we will use to find the significance level.

In the online phase, when an insertion happens, we take a sample of the new data, S_t and use the latest model, M_{t-1}, to calculate the average likelihood of S_t. Finally, we compare the test statistic d with the threshold ($2 \times std$). Given the normality of the above distribution of average likelihoods, we know that if one were to draw another sample $S_{t-1}^{\leq}$ from the previous data its average likelihood would fall within $2 \times std$ from the mean with probability 95%. Now, if $d > 2 \times std$ we conclude that we are not confident enough to accept that the new data has the same distribution of the old data – that is, we reject the null hypothesis with a p-value of 0.05.

3.5 The Test Errors

There are two errors associated with a hypothesis testing. *type-1 error* is rejecting the null hypothesis when it should not. *Type-2 error* is the error of accepting the null hypothesis when it should be rejected. The first one introduces false positives to the system and the second causes false negatives. False positives (FPs) are only a (rather small) performance concern only, spending time to update the model while accuracy is preserved. False negatives (FNs), however, can cause a loss of accuracy. Therefore, the system can afford to be stricter with respect to the significance level, in order to reduce the risk of false negatives and accuracy loss.

DDUp uses the loss of the trained NNs for OOD detection. Sometimes NNs could be over-confident [48, 51, 57] which may introduce bias. However, we have not witnessed it for our tasks here on tabular data. If there were bias, the FP and FN rates discussed above would signal it. We have evaluated DDUp with respect to FPs/FNs in Section 5.2 showing that this is not a concern.

4 MODEL UPDATE

In this section, we propose a transfer-learning based method that can retain previous knowledge of the model while adapt it to the new insertions. The OOD detection module will either output 'in-distribution' or 'out-of-distribution' signals.

The in-distribution case. When no drift occurs, the new data distribution is similar to that of the historical data and this distribution could be represented by a similar parameter space of the latest model, M_t. Hence, the learned component of the system could remain unchanged. More specifically, the framework can copy M_t to M_{t+1} and update the required meta-data associated with the system (such as the frequency tables in DBEst++, or table cardinalities in Naru/NeuroCard). Even if there are slight permutations in data, fine-tuning the latest model's parameters on the new data will adjust it to the general representation of both old and new data. We will show that when knowing that data is not OOD, *fine-tuning* with a relatively small learning rate, can retain model performance. Specifically, with an **in-distribution** signal at time $t+1$, M_t is retrained on S_{t+1} with a small learning rate, lr. This learning rate could be tuned, as a hyper-parameter. We intuitively set $lr_t = \frac{|D_{t+1}|}{|D_t^{\leq}|} \times lr_0$ and experimentally show that it is a good choice.

The OOD case. With a distributional shift, by fine-tuning on new data, the model's parameters would bias toward the new data distribution. Even smaller learning rates cause tiny deviations from the previous parameter space which may yield large errors during inference. And, retraining using all the data from scratch is too time consuming. Thus, we propose an updating approach grounded on the transfer-learning paradigm. The general idea is to use the learned model M_t and incorporate it in training M_{t+1}. To this end, we utilize the *knowledge distillation* principles, which help to transfer the previously learned knowledge to a new model. Our rationale for such a model updating approach is based on the following:

- Distillation has several benefits including: faster optimization, better generalization, and may even outperform the directly trained models. [79].
- It is accurate for queries on old as well as new data.
- It allows us to control the weights for queries on new and old data with just a couple of parameters.
- It is efficient memory-wise as well as computationally-wise, compared to methods like Gradient Episodic Memory, or Elastic Weight Consolidation and PathInt (cf. Section 6)
- It does not make any assumptions about the training of the underlying models. This property, is especially desirable since: a) we can use it to update different neural networks; b) it prevents the high costs of rebuilding base models; c) different pre-processings could be left intact. For instance, Naru, DBEst++ and TVAE all use completely different types of embedding/encoding. DDUp can update the model regardless of these differences.

4.1 General Knowledge Distillation (KD)

KD was first introduced in [22] for *model compression* by transferring knowledge from an accurate and "cumbersome" model, called *teacher*, to a smaller model called *student*. In its basic form, instead of fitting the student model directly to the actual data *labels*, one would use the class probability distribution learned by the teacher to fit the student model. Hinton et al. [22] argued that small probabilities in "wrong" label logits, known as "soft labels", include extra information called "dark knowledge" that result in better learning than actual "hard labels". Distillation has since been extensively studied. Figure 2 shows a general view of the principles of a distillation process. A small dataset referred to as *transfer-set* is fed into a pre-trained model (teacher) and a new model (student) to be trained. A *distillation loss* is calculated using the predictions of the pre-trained model instead of the actual labels. This loss and a typical loss using actual labels will be used to train the new model.

To formulate *knowledge distillation*, consider a model with parameters Θ, representing a function f_t (t for teacher) which has been trained via Eq. 2. We would like to transfer knowledge from this teacher model to a student model with parameter Θ', representing a function f_s. This new model could be trained as follows:

$$f_{s\Theta'} = \underset{f\in\mathcal{F}}{\arg\min}\ \frac{1}{|tr|}\sum_{i\in tr}\left[\lambda\mathscr{L}_d(f_s(i);f_t(i);\Theta;\Theta') + (1-\lambda)\mathscr{L}(f_s(i);\Theta')\right] \tag{4}$$

for weight λ, distillation loss $\mathscr{L}_d$, and transfer-set tr.

4.2 DDUp: Updating By Knowledge Distillation

[12, 66] showed that, for classification tasks, if instead of having a compact student model, one uses the same architecture of the teacher, and repeat distillation sequentially for several generations, the student models in the later generations could outperform the teacher model. This approach is called *sequential self-distillation*. Inspired by this and anticipating that this will be valid for our learning tasks, DDUp also employs a sequential self-distillation approach.

To update a model using KD, a copy of the previously trained model becomes the new student. Then, the student is updated using a distillation loss (to be defined soon). After updating, the previous teacher is replaced with the new updated model. This cycle repeats with every new insertion batch.

To formulate our training loss function, we consider two aspects that we would like to have in our updating scheme. First, to have control over the the new data/queries versus the old data/queries. Second, to make it general so that different learned DB systems could adopt it. As such, we first write down the general form of the total loss function and then, use cross-entropy and mean-squared-error as the loss functions to instantiate different models. Training in each update step is as follows:

$$\begin{aligned} f_{s\Theta'} = \underset{f\in\mathcal{F}}{\arg\min}\Bigg(&\alpha \times \frac{1}{|tr|}\sum_{x\in tr}\Big[\lambda\mathscr{L}_d(f_s(x), f_t(x);\Theta') \\ &+ (1-\lambda)\mathscr{L}(f_s(x);\Theta')\Big] \\ &+ (1-\alpha)\times\frac{1}{|up|}\sum_{x\in up}\mathscr{L}(f_s(x);\Theta')\Bigg) \end{aligned} \tag{5}$$

Here, α and λ are the new data and the distillation weights, respectively. Also, tr and up are the transfer-set and the update batch. In summary, the rationale for proposing this novel loss function is: The transfer-set term acts as a regularizer to avoid overfitting on new

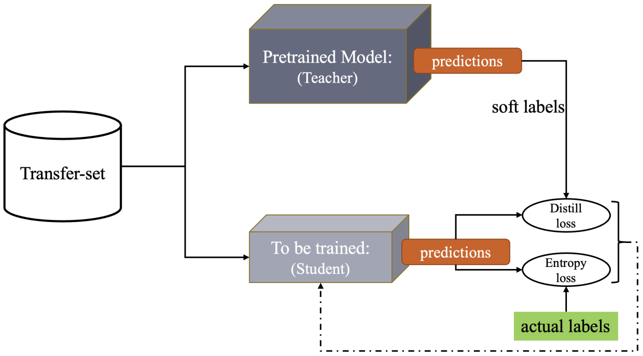


Figure 2: The knowledge distillation process.

data. The same goal is also helped by self-distillation (when copying the teacher to the student). Additionally, as mentioned sequential self-distillation [66] may attain increasingly higher accuracy, even outperforming "retrain from scratch" (cf. Section 5.3).

For models that provide a conditional probability in the last layer of the network (e.g. using a Softmax function), an annealed cross-entropy loss will be employed. Otherwise, we utilize mean-squared-error using the logits from the last layer of the network. Eq. 6 and Eq. 7 show these two loss functions.

$$\mathcal{L}_{ce}(D_{tr}; z_t, z_s) = -\sum_{i \in [k]} \frac{exp(z_{t_i}/T)}{\sum_{j \in [k]} exp(z_{t_j}/T)} \log \frac{exp(z_{s_i}/T)}{\sum_{j \in [k]} exp(z_{s_j}/T)} \tag{6}$$

$$\mathcal{L}_{mse}(D_{tr}; z_t, z_s) = \sum_{i \in [|z_t|]} (z_{t_i} - z_{s_i})^2 \tag{7}$$

where D_{tr} is the *transfer-set* , T is a temperature scalar to smooth the probabilities so that it produces "softer" targets, and $[k]$ is the vector $[0, 1, \ldots, n]$ which are the class probabilities and $[|z_t|]$ indicates the logits of the network.

4.3 Instantiating the Approach

Mixture Density Networks. MDNs consist of an NN to learn feature vectors and a mixture model to learn the *probability density function* (pdf) of data. Ma et al. [42] uses MDNs with Gaussian nodes to perform AQP. For the Gaussian Mixture, the last layer of MDN consists of three sets of nodes $\{\omega_i, \mu_i, \sigma_i\}_{i=1}^{m}$ that form the pdf according to Eq. 8.

$$\hat{P}(y|x_1, ..., x_n) = \sum_{i=1}^{m} \omega_i . \mathcal{N}(\mu_i, \sigma_i) \tag{8}$$

where m is the number of Gaussian components, y is the dependent variable and $(x_1, ..., x_n)$ is a set of independent variables, w_i is the weight of the i^{th} Gaussian with a mean of μ_i and a standard deviation of σ_i. For MDNs, we define distillation loss as follows:

$$\mathcal{L}_d = \mathcal{L}_{ce}(D_{tr}, \omega_t, \omega_s) + \mathcal{L}_{mse}(D_{tr}, \mu_t, \mu_s) + \mathcal{L}_{mse}(D_{tr}, \sigma_t, \sigma_s) \tag{9}$$

This summation of terms help us retain both the shape of data distribution as well as the intensity levels.

Deep Autoregressive Networks. The Naru and NeuroCard cardinality estimators [77, 78] use deep autoregressive networks (DARNs) to approximate a fully factorized data density. DARNs are generative models capable of learning full conditional probabilities of a sequence using a masked autoencoder via Maximum Likelihood. Once the conditionals are available, the joint data distribution could be represented by the product rule as follows:

$$\hat{P}(A_1, A_2, \ldots, A_n) = \hat{P}(A_1)\hat{P}(A_2|A_1)\ldots\hat{P}(A_n|A1, \ldots, A_{n-1})$$

where A_i is an attribute in a relation R. Naru and NeuroCard use cross-entropy between input and conditionals as the loss function. This allows us to formulate the distillation loss function using the conditionals of the teacher and the student networks. Also, in Naru and NeuroCard, each conditional is calculated using a set of logits, hence we average over all as follows:

$$\mathcal{L}_d = \frac{1}{|A|} \sum_{i=1}^{|A|} \mathcal{L}_{ce}(D_{tr}, z_{s_i}, z_{t_i}) \tag{10}$$

Where $|A|$ is the number of attributes corresponding to the number of conditionals.

Variational Autoencoders. VAEs have been used for a number of DB components: [65] for AQP, [17] for CE, and [76] for synthetic tabular data generation. They are a type of autoencoders that instead of learning deterministic encoder, decoder, and compressed vector (known as bottleneck), they learn a probabilistic encoder, decoder, and a latent random variable instead of the compressed vectors. (For more details, see the seminal paper [26]). Interestingly, a VAE is trained using a different loss function, known as Evidence-Lower-Bound (ELBO) loss (which amounts to a lower bound estimation of the likelihoods). Here we shall use TVAE for learned synthetic tabular data generation (of particular importance in privacy-sensitive environments, or when data is scarce for data augmentation purposes, or when wishing to train models over tables and accessing raw data is expensive in terms of time or money).

To distill a VAE, one must cope with the random noise added to the input of the decoder by the latent variable. For that, the latent variable in the teacher network is removed, and we use the same noise generated by the student in the teacher. The reason for doing this is that distillation tries to teach the student to behave like the teacher for a specific observation or action. If there is randomness, the student might mimic the teacher's behaviour for a completely different observation. After this change, the corresponding logits of the encoder/encoder of the student and the teacher are compared using MSE. Finally, the loss function is:

$$\mathcal{L}_d = \frac{1}{2}(\mathcal{L}_{mse}(D_{tr}, z_t^{(e)}, z_s^{(e)}) + \mathcal{L}_{mse}(D_{tr}, z_t^{(d)}, z_s^{(d)})) \tag{11}$$

where, e and d correspond to the encoder and the decoder networks.

4.4 An Example

We create a simple synthetic dataset consisting of a categorical attribute, x, with 10 *distinct-values* $= \{1, 2, 3, \ldots, 9, 10\}$, and with each category having 1000 real values. The dataset is balanced and the real values for each category are generated by a *Mixture of Gaussians* (MoG) with five peaks. Figure 3.a is the dataset corresponding to $x = 1$. We fit a *Mixture Density Network* with ten components on this dataset. Figure 3.b shows a sample generated by this MDN which asserts that the model has perfectly learnt the data distribution. Next, we introduce an update batch generated by a MoG with two different means. Figure 3.c shows the update batches in red color compared to the previous data in blue. We update the previously learned MDN with the proposed loss function in Eq. 9. We repeat updates for 50 batches generated with the new MoG. Figure 3.d shows the final distribution learnt by the MDN.

4.5 Handling Join Operations

DDUp can operate either on raw tables or tables from join results. If the old data R is the result of a join, the new data batch needs to be computed, due to new tuples being inserted in any of the joined

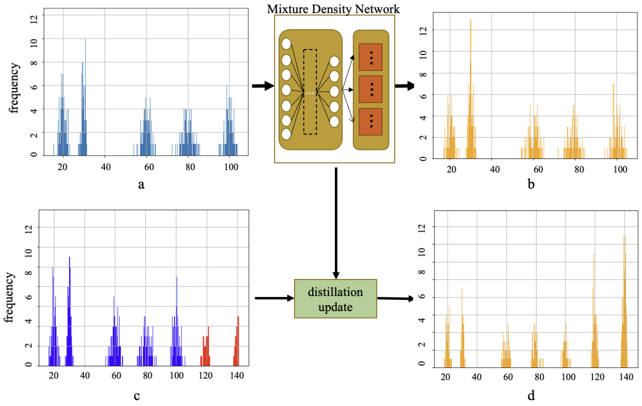


Figure 3: An example to show how DDUp learns new data without forgetting. 'a' is the histogram of synthetic data corresponding to $x = 1$. 'b' is the sample generated by the learned MDN for $x = 1$. 'c' shows a sample of an update batch coming from different Gaussians. 'd' is the sample generated by the MDN after being updated by the DDUp loss function. We have performed the update 50 times to see the effect of high frequency updates (This explains the higher frequencies around the last two peaks for 'd').

tables in R. Consider at time $t-1$ a model M_{t-1} has been trained on $R = \bigcup_{j=0}^{t-1} T_1^j \bowtie \bigcup_{j=0}^{t-1} T_2^j \ldots \bowtie \bigcup_{j=0}^{t-1} T_n^j$, where T_r^j denotes the new batch for table T_r at time j. Without loss of generality, suppose a new insertion operation I_t at time t adds new data to table T_i, denoted T_i^t. The new data for DDUp in this setting is $D_t = (R \setminus \bigcup_{j=0}^{t-1} T_i^j) \bowtie T_i^t$, where $\setminus$ denotes a (multi)set-difference operator. Therefore, for the detection module, $S_{t-1}^{\leq}$ is a sample of R, and S_t a sample from D_t. Furthermore, during updating the transfer-set is a sample from R and the new data is D_t. Please note that all this data preparation and how each model deals with joins is orthogonal to DDUp. Therefore, it can be done by either computing the actual joins above or using join samplers like [62, 82], as is done in NeuroCard and compared against in Section 5.4.

5 EXPERIMENTAL EVALUATION

We evaluate DDUp for three different models for learned DB components: (i) Naru/NeuroCard [77, 78] which use DARN models for CE; (ii) DBest++ [42] that uses MDNs for AQP; and (iii) TVAE [76], that uses variational autoencoders for DG. We evaluate in terms of model accuracy and update time. We use as reference points the baseline update approach provided for AQP and CE (TVAE provides no update approach). We also add as reference points the accuracy when retraining from scratch and when leaving models stale. With respect to *OOD detection*, we investigate whether it can detect significant data shifts successfully and how this will contribute to the final performance of the underlying models in their specific application, CE, AQP, DG. Ultimately, the experiments are to address the following questions:

- How to best evaluate DDUp? (Section 5.1)
- Can DDUp accurately detect a distributional shift? (Section 5.2)
- Is DDUp accurate under in- *and* out-of- distribution settings? (Section 5.3)
- How does DDUp compare to the baseline approaches in accuracy and update time? (Section 5.3)
- What is the effect of distillation? (Section 5.5)
- Is DDUp efficient? (Section 5.6)

5.1 Experimental Setup

To establish a dynamic setup, we make a copy of the base table and randomly sample 20% of its rows as new data. In this setting, new data follows the previous data distribution which we denote as *in-distribution*. We introduce distributional drift as is typically done for tabular data settings, say in [70]. As such, after making the copy, we sort every column of the copied table individually in-place to permute the joint distribution of attributes. Next, we shuffle the rows and randomly select 20% of the rows - this now becomes the new data. With these new data, we perform two types of experiments. First, we consider the whole 20% sample as a new data batch and update the model with it. Second, to show the updatability in incremental steps, we split the 20% data into 5 batches. In general, the size of the transfer-set is a tunable parameter [22], influenced by the dataset complexity, the underlying model generalization ability, and the downstream tasks. After tuning, we used a 10% transfer-set for MDN and DARN and a 5% for TVAE, which could be further tuned with methods like Grid search.

DDUp does not impose any further constraints to those of the underlying models. For DBest++ we use a query template with a range and an equality attribute. Also, we use one-hot encoding to encode categorical attributes and normalize the range attribute to $[-1, 1]$. For Naru/NeuroCard and TVAE, we use the same settings as explained in their code documentation. We use the learned hyperparameters of the base model, i.e the model we build at time zero, for all subsequent updates. Furthermore, we intuitively set α parameter in Eq. 5 to the fraction of update batch size to the original data size and tune λ for values in $[9/10, 5/6, 1/4, 1/2]$.

5.1.1 *Datasets.*

We have mainly used three real-world datasets (census, forest, DMV) (see Table 1). These datasets have been widely used in the learned DB literature. For CE, [70] uses also forest, census and DMV, while NeuroCard/Naru use JOB/DMV. For AQP DBEst++ uses TPCDS. For DG, [76] uses census and forest. Thus, we have also used census, forest, DMV, and TPCDS (`store sales` table, scaling factor of 1). Finally, for join queries, we have used JOB (on IMDB data) and TPCH benchmarks, which are also used in [77, 78].

Table 1: Characteristics of datasets.

Dataset	Rows	Columns	Joint Domain
Census	49K	13	10^{16}
Forest	581K	10	10^{27}
DMV	11.6M	11	10^{15}
TPCDS	1M	7	10^{30}

5.1.2 *Workload.*

Each model is evaluated using 2,000 randomly generated queries. These queries are generated at time zero for each model and are used throughout the subsequent updates. When an update batch is performed, the ground truth of the queries will be updated. For Naru/NeuroCard, we use their generator to synthesize queries: It randomly selects the number of filters per query (forest:[3,8], census: [5,12], TPCDS: [2,6], dmv: [5,12]). Then, it uniformly selects a row of the table and randomly assigns operators [=, >=, <=] to the columns corresponding to the selected filters. Columns with a domain less than 10 are considered categorical and only equality filters are used for them. For DBest++, we select a *lower-bound* and a *higher-bound* for the range filter and uniformly select a category from the categorical column for the equality filter. Throughout the experiments, we discard queries with actual zero answer. The structure of a typical query in our experiments is:

SELECT AGG(y) FROM $T_1 \bowtie T_2 \ldots \bowtie T_n$ WHERE F_1 AND ... AND F_d

where, F_i is a filter in one of these forms: $[att_i = val, att_i >= val, att_i <= val]$. Also, `AGG` is an aggregation function like `COUNT`, `SUM`, `AVG`. For DBest++, the query template contains one categorical attribute and one range attribute. As such, we select the following columns from each dataset: census:[`age, country`]; forest:[`slope, elevation`]; dmv:[`body type, max gross weight`]; TPCDS:[`ss quantity,ss sales price`]; IMDB:[`info type id,production year`]; TPCH:[`order date,total price`] where the first/second attribute is categorical/numeric. Furthermore, Naru could not train on the full TPCDS dataset as the encodings were too large to fit to memory. Hence, we selected the following columns [`ss sold date sk`, `ss item sk`, `ss customer sk`,`ss store sk`, `ss quantity`, `ss net profit`], and made a 500k sample.

5.1.3 Metrics. For *count* queries, we use *q-error* as follows:

$$error = \frac{max(pred(q), real(q))}{min(pred(q), real(q))} \tag{12}$$

For *sum* and *avg* aggregates, we use *relative-error* as follows:

$$error = \frac{|pred(q) - real(q)|}{real(q)} \times 100 \tag{13}$$

Additionally, Lopez et al. [38] introduce the notions of Backward Transfer (BWT) and Forward Transfer (FWT) as new metrics in class incremental learning tasks. BWT is the average accuracy of the model on old tasks, and FWT is the average accuracy of the model on new tasks. Here, we re-frame BWT and FWT. We generate the queries at time 0 and use them for all update steps. At each step t, we calculate $diff = real_t(q) - real_{t-1}(q)$ for each query, q, which gives us three set of queries; G_{fix} with $diff = 0$, $G_{changed}$ with $diff > 0$, and $G_{all} = G_{fix} \cup G_{changed}$. With these groups, we define three measures. AT: average q-error over G_{all}. FWT: average q-error over $G_{changed}$. BWT: average q-error over G_{fix}.

5.1.4 Evaluating Variational Autoencoders. DG is an interesting learned application which is recently supported using TVAE. Thus, we evaluate DDUp for TVAE. In TVAE, once the training is done, only the decoder network is kept and used, as this is the generator. Hence, we apply our distillation-update method to the decoder network. We evaluate TVAE via the accuracy of an XGboost classifier trained by the synthetic samples, as in [76]. We hold-out 30% of table as the test set, and train two classifiers with original and synthetic data, then predict the classes of the held-out data. We report *micro f1-score* for classifiers. For census, forest and DMV, we use: *income*, *cover-type*, and *fuel-type*, as the target class, respectively. For TVAE, we created a smaller DMV with 1m records, as training TVAE on the whole DMV is very time/resource consuming (proving indirectly the need to avoid retraining).

5.2 OOD Detection

5.2.1 Loss Functions as Signals. We first show the results of loss/log-likelihoods when the detector receives samples from the same distributions or from different distributions. The results are shown in Table 2. For Naru/NeuroCard and DBEst++ we report the actual log-likelihood values (not negatives, so higher is better). For TVAE, we report the ELBO loss values (hence lower is better).

Table 2: Average log-likelihood and ELBO loss values of data samples on a trained model. S_{old} is a sample of the previous training data. "IND", is a 20% sample from a straight copy of the original table; "OOD", is a 20% sample from a permuted copy of the original table.

Dataset	DBEst++			Naru/NeuroCard			TVAE		
	S_{old}	IND	OOD	S_{old}	IND	OOD	S_{old}	IND	OOD
Census	-0.362	-0.361	-0.366	-20.99	-20.87	-36.95	-15.21	-15.22	81.47
Forest	-0.0194	-0.0202	-0.052	-43.16	-43.9	-141.10	-19.96	-20.09	142.38
DMV	2.520	2.532	2.444	-13.74	-13.16	-18.67	9.114	9.28	34.95

Table 2 shows that the loss function (log likelihood and ELBO in our cases) can reliably signal OOD data. Interestingly, this corroborates similar findings in [18] for classification tasks in various vision and NLP tasks, where the NN outputs can be used to signal OOD. Here we show it for tabular data and for NNs developed for AQP, CE, and DG tasks.

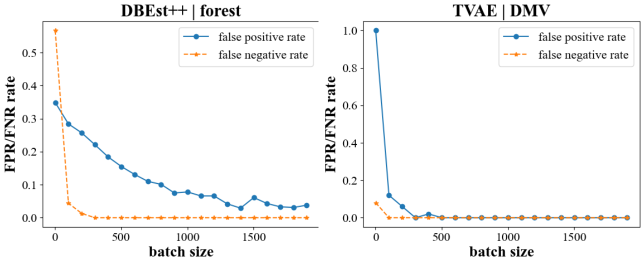
In Naru/NeuroCard and TVAE, when permuting, all columns are sorted individually, hence the large difference in likelihoods. For DBEst++, only the selected columns for a query template have been permuted, yielding a small difference in likelihoods.

5.2.2 The two-sample test results. Table 3 shows results for two-sample testing for OOD detection. The significance level of the test (threshold) is $2 \times std$ of the bootstrapping distribution, which was obtained by >1000 iterations. In each iteration, we use a 1% sample with replacement from previous data and a 10% sample without replacement from new data to calculate the test statistic. The results show that when data is permuted, the test statistic is far away from the threshold. This means it appears at a great dissonance in the tails of the bootstrapping distribution. And since the critical value to test for OOD is found by bootstrapping over S_{old}, i.e., $S_t^{\leq}$, it will adjust even to small differences when faced with OOD. Case in point, the DBEst++ OOD likelihood value for census (which is similar to IND/S_{old} in Table 2) vs the corresponding test-statistic value in Table 3.

5.2.3 FP and FN rates in OOD detection. To evaluate OOD detection, we measure FP and FN rates (FPR, FNR). We created an OOD test-set and an IND test-set, each equaling half the original size of the table. The latter is just a random sample from the original table. The former is constructed as follows. The perturbed data is obtained by perturbing one or more of five columns of the table, say $C1$, ... $C5$. First we perturb $C1$ and take a sample of the resulting table of size 10% and append it to the OOD test-set. Then we perturb $C1$ and $C2$ and similarly sample and append it to the OOD test-set. We repeat this for perturbations on $C1, C2, C3$, on $C1, C2, C3, C4$, and on $C1, C2, C3, C4, C5$, ending up with an OOD test-set of size 50% of the original table. Note that this setup creates a more-challenging case, as the degree of perturbations (for OOD

Table 3: The test-statistic values. Threshold is $2 \times standard - deviation$ and bs-mean is the mean of bootstrapping distribution.

Dataset	DBEst++				Naru/NeuroCard				TVAE			
	bs-mean	threshold	IND	OOD	bs-mean	threshold	IND	OOD	bs-mean	threshold	IND	OOD
Census	-0.3524	0.007	0.001	0.05	-21.0076	0.0529	0.032	16.0052	-15.1834	0.6041	0.0419	100.5126
Forest	-0.0228	0.0122	0.007	0.2315	-41.35	0.0141	0.0084	72.5473	-19.99	0.0868	0.0417	167.0502
DMV	2.52	0.1287	0.0145	4.5745	-13.7674	0.0012	0.0007	5.1145	9.1209	0.0177	0.0015	25.1398



(a) (b)

Figure 4: Sensitivity of OOD detection vs batch size.

data) is finer-grained. Then, at each batch, we fed a random sample from the OOD test-set and of the IND test-set to the DDUp detector. For each batch, the detector would signal IND or OOD and we recorded and calculated FPR and FNR. The batch size was 2,000 and we repeated the experiment for 1,000 batches.

We used the same parameters for all datasets and models: the bootstrapping size is 32 and the threshold is $2{\times}std$. For DBEst++, the results are reported in Table 4. FPR and FNR for Naru/NeuroCard and TVAE were always zero. These results further confirm that the OOD detection algorithm is not biased.

Table 4: FPR and FNR for DBEst++.

Dataset	FPR	FNR
Census	0.15	0.01
Forest	0.10	0
DMV	0.01	0

Furthermore, we studied the sensitivity on the batch size and varied it from a size of 1 to 2,000. Results are shown in Figure 4, which clearly show that after a low-threshold batch size, FPR and FPN tend to zero. The same results hold for other models and datasets, and are omitted here for space reasons.

5.3 Accuracy Results

5.3.1 When there is OOD data. For Naru/NeuroCard, DBEst++, and TVAE, and for each dataset, we compare 4 updating approaches against each other and against the base model before any new data is inserted. The 4 approaches are as follows: "`Retrain`", retrains the model from scratch using both old and new data. "`Baseline`" is the baseline approach in Naru/NeuroCard and DBest++ where a trained model is updated with new data by performing *SGD* with a smaller learning rate. "`DDUp`" is the proposed method. Finally, in "`stale`", the model is not updated – this is a do-nothing approach. For reference, we also include the numbers for M_0, i.e., the original model accuracy before any new data came. Table 5 and Table 6 show the accuracy results for CE and AQP (SUM and AVG operations), respectively. For TVAE, the classification f1-scores are reported in Table 7. Results of these three tables correspond to the case where the update sample is permuted. DDUp always performs better than the baseline approach. Most of the times, the performance of DBEst++ on DMV dataset is not as well as for the other datasets. This probably is due to the complexity of data (large scale and highly correlated attributes). Nevertheless, DDUp stands on the top of the underlying models and regardless of the model's performance, DDUp ensures that it will retain the accuracy. Please note the DMV dataset results in Table 5 and Table 6 and, census and forest datasets in Table 7, where, DDUp even outperforms retraining from scratch. Interestingly, this corroborates similar evidence for sequential self-distillation (for boosting embeddings for) classification tasks [66]. This was one of the reasons we adapted a self-distillation based approach. Finally, baseline methods have poor performance for 95th and 99th percentiles.

Performance on old and new queries. To better illustrate the effects of *catastrophic forgetting* and *intransigence* we elaborate on performance on FWT and BWT. (As `retrain` avoids be definition *catastrophic forgetting* and *intransigence*, it is omitted). The results are shown in Table 8. Note that any insertion affects only a percentage of queries, shown in Table 9. Comparing AT, FWT, and BWT in Table 5 and Table 8 first note that fine-tuning always performs much better in terms of FWT compared to BWT (due to catastrophic forgetting). Second, conversely, a stale model shows better BWT compared to FWT. For DDUp, FWT and BWT remain close to each other, especially in terms of median q-error, showing that DDUP can ensure accuracy for queries on old and new data. Overall, DDUp enjoys high accuracy.

Incremental Steps. To show the updates in incremental steps, we have split the 20% data into 5 equal-sized chunks and have performed an update incrementally for each batch. Figure 5 compares the trend of accuracy during updates. As it is clear from the figures, DDUp remains very close to `retrain`, while there is a drastic drop in accuracy using `baseline`. Starting point 0 is where the base model M_0 is built from scratch. (The same results hold for 95th, 99th percentiles and maximum q-error).

We also have evaluated the models with respect to the *log-likelihood goodness-of-fit.* Log-likelihood is widely used to evaluate NN models. Using log-likelihood allows evaluation to be independent of underlying applications. Figure 6 shows changes in log-likelihood in consecutive update steps. At each step, we calculate the average of log-likelihoods over a sample of new data and a sample from historical data. In these figures we again see that updating with DDUp is fitting to the old and the new data very similarly to the `retrain` case. In general, when keep using `stale`,

Table 5: Results of updating a base model with a 20% permuted sample in terms of q-error. M_0 denotes the base model.

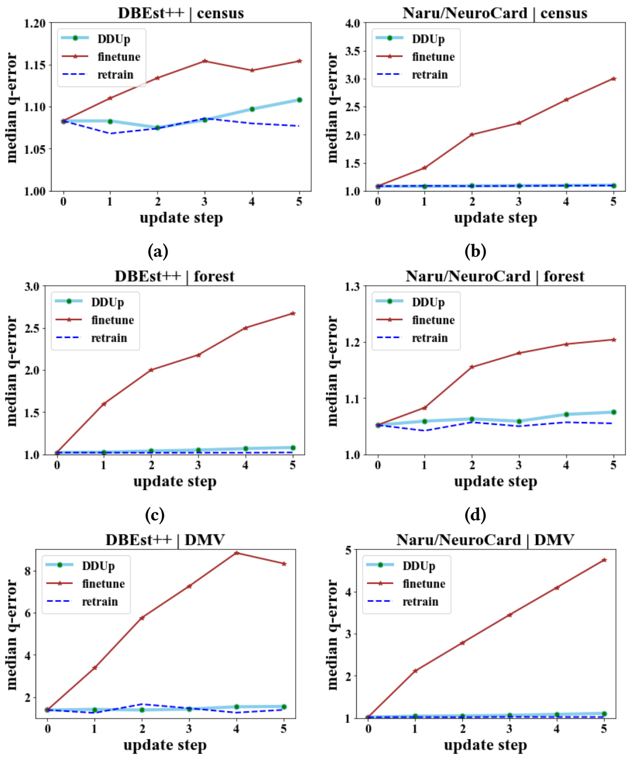
Dataset	metric	DBEst++					Naru/NeuroCard				
		M_0	DDUp	baseline	stale	retrain	M_0	DDUp	baseline	stale	retrain
census	median	1.05	1.11	1.17	1.16	1.07	1.08	1.09	4	1.14	1.07
	95th	2	2	2.20	2	2	2	2	471.80	2	2
	99th	3	3	4	3	3	3	3	1534.69	3.16	3
	max	5	7	11	10.50	5	5.25	7	8385	21.88	6
forest	median	1.026	1.046	2	1.18	1.02	1.04	1.07	1.54	1.10	1.05
	95th	2	2	63.40	2	1.64	2.48	3	41	2.50	2.75
	99th	2	2.583	503.12	5.60	2	4	6	157.16	5.48	5
	max	4	5.33	3470	90.85	5.33	27	65.66	1691	484	34.66
DMV	median	1.20	1.143	3.48	1.88	1.34	1.02	1.04	2.57	1.16	1.02
	95th	4.91	5.07	234.88	7.00	5.50	1.20	1.41	468.68	1.50	1.25
	99th	9.65	10	3897.87	12.50	8	1.83	2.31	4734.62	2.84	2
	max	18.83	19	65875	39	17	8	9.81	343761	9.49	5
TPCDS	median	1.02	1.04	57	1.27	1.02	1.01	1.07	1.15	1.10	1.05
	95th	1.16	1.26	269	1.58	1.18	2	2	29	2	2
	99th	1.5	1.61	1266	2.72	1.5	3.01	3.01	239	4	3
	max	3	3	4534	10.66	5.64	5	28	5100	28	24

Table 6: mean-relative-error for SUM and AVG aggregation functions for DBEst++.

Dataset	function	M_0	DDUp	baseline	stale	retrain
census	SUM	13.05	17.30	65.88	21.36	13.60
	AVG	1.89	2.36	8.15	2.37	1.97
forest	SUM	10.11	15.51	88.73	24.59	10.14
	AVG	0.76	1.04	3.90	1.35	0.79
TPCDS	SUM	4.53	6.37	61.40	22.64	5.12
	AVG	0.88	1.47	12	3.50	1.21
DMV	SUM	76.73	85.29	423	97.00	110
	AVG	6.4	6.9	15.9	8.6	7.3

Table 7: Classification results for TVAE in terms of micro f1. 'r' stands for real data, 's' stands for synthetic data.

Dataset	M_0		DDUp		baseline		stale		retrain	
	r	s	r	s	r	s	r	s	r	s
census	0.67	0.63	0.77	0.73	0.77	0.55	0.77	0.56	0.77	0.72
forest	0.84	0.69	0.89	0.78	0.89	0.63	0.89	0.60	0.89	0.74
DMV	0.97	0.97	0.98	0.97	0.98	0.92	0.98	0.93	0.98	0.98



(a)

(b)

(c)

(d)

(e)

(f)

Figure 5: Updating results over 5 consecutive updates.

the log-likelihood drops after the first update and then remains low. The reason is that all update batches have similar permutation and since we calculate unweighted averages, the log-likelihood stays fixed. While, for `baseline`, i.e fine-tuning, we can see a gradual decrease of likelihood which means that the network is increasingly forgetting about previous data in each step.

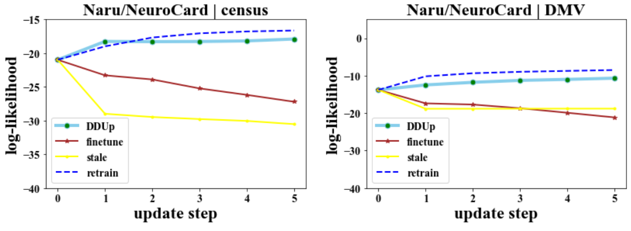
5.3.2 When data is not OOD. In this case, simple fine-tuning update algorithms, such as `baseline`, will likely avoid *catastrophic forgetting*. To illustrate this, we have repeated the 5 batched incremental updates with data without permutation. The results are reported in Figure 7. For space reasons, we only show the results

Table 8: Comparing q-error of different updating approaches in terms of FWT and BWT.

Dataset	metric	DBEst++							Naru/NeuroCard						
		M_0	DDUp		baseline		stale		M_0	DDUp		baseline		stale	
			FWT	BWT	FWT	BWT	FWT	BWT		FWT	BWT	FWT	BWT	FWT	BWT
census	median	1.05	1.06	1.12	1.06	1.20	1.05	1.16	1.08	1.11	1.09	1.83	6	1.20	1.13
	95th	2	1.66	2	1.56	2.33	3.30	2	2	1.64	2	4.63	530.80	3.18	2
	99th	3	4.94	3	4.10	4	8.90	2.75	3	3.08	3	9.98	1598.53	8.49	3
forest	median	1.02	1.01	1.08	1.23	2.66	1.05	1.20	1.04	1.07	1.07	1.39	1.65	1.18	1.08
	95th	2	1.181	2	2.87	146.38	2.85	2	2.489	1.88	3	3.13	43.02	7.55	2.33
	99th	2	1.52	3	3.72	590.57	18.33	2.24	4	4.89	6	5.27	163.80	191.53	4.86
DMV	median	1.20	1.28	1.13	2.20	4.36	1.66	1.54	1.02	1.02	1.07	1.06	12.85	1.26	1.19
	95th	4.910	4.30	5.87	3.34	484.46	9.50	6.87	1.20	1.16	1.55	1.65	1015.81	3.30	1.40
	99th	9.65	9	11.65	10.50	5894.21	12.12	10.80	1.83	1.47	3	3.35	8183.34	11.93	2.49
TPCDS	median	1.02	1.03	1.04	1.20	1.51	1.16	1.21	1.01	1.06	1.08	1.19	1.11	1.10	1.10
	95th	1.16	1.21	1.29	2.37	339	2.26	1.35	2	2	2	2.60	54	2	2
	99th	1.5	1.37	1.66	4.27	1536	4.48	1.66	3.01	9.77	3	9.47	434	9.64	3.77

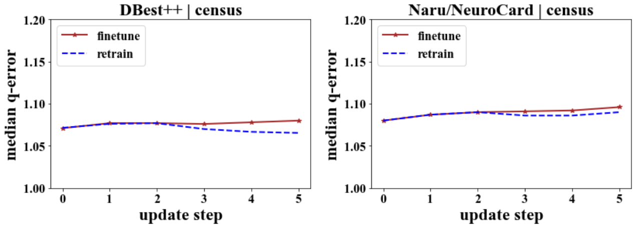
Table 9: The percentage of the queries (out of 2k queries) with changed actual results after inserting 20% new data.

dataset	DBEst++	Naru
census	14%	12%
forest	32%	9%
TPCDS	36%	36%
dmv	52%	45%



(a) (b)

Figure 6: log-likelihood results over 5 consecutive updates.



(a) (b)

Figure 7: Updating results over 5 consecutive updates when data follows the same distribution as the historical data.

for census. The results indicate that for in-distribution data, simple baselines can have a performance close to `retrain`.

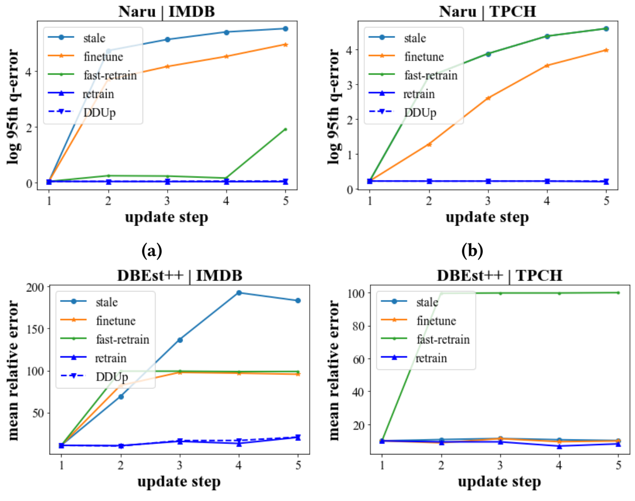
5.4 Evaluating DDUp for Join Queries

As mentioned, DDUp is unconcerned whether at a time t, $S_{t-1}^{\leq}$ (a sample of $\cup_{j=0}^{t-1} D_j$) and D_t come from a raw table or from a join. For this experiment, we have evaluated DDUp running 2,000 queries over two 3-table joins from the JOB and TPCH datasets. For each, the 2,000 queries involve a join of the fact table with two dimension tables: Specifically, the join of tables [`title`, `movie info idx`, `movie companies`] for IMDB, and [`orders`, `customer`, `nation`] for TPCH. For the update dynamics, we have split the fact table into 5 time-ordered equally-sized partitions. We have built M_0 on the join (of the fact table's first partition with the 2 dimension tables) and updated it with each subsequent partition at a time. This is similar to the update setting in NeuroCard. Results for both CE and AQP are in Figure 8.

NeuroCard, unlike other models, natively supports joins, using a "fast-retrain" - i.e., a light retraining where the model is retrained using a 1 percent sample of the full join result. We have included this policy here as "fast-retrain". DDUp always signalled OOD for the new update batches, except for TPCH data on DBest++, where update was not triggered. Therefore, in Figure 8.d the accuracy of the stale model and fine-tuning is close to retrain. This further confirms the significance of OOD detection.

Table 10: DDUp's speed up over `retrain`, for two update sizes. For census, forest, and dmv, sp1: 20% of the original table. sp2, 5% of the original table. for IMDB and TPCH sp1: updating the first partition and sp2: updating the last partition.

Dataset	DBEst++		Naru		TVAE	
	sp1	sp2	sp1	sp2	sp1	sp2
census	5	5.5	3.5	4	3.4	5.7
forest	1.6	4	5	9.2	3.6	7
DMV	4	6.5	2.3	9.6	3.4	6.8
IMDB	4.5	18	3.5	5	NA	NA
tpch	6.5	16	2	4	NA	NA



(a) (b) (c) (d)

Figure 8: DDUp's performance on joined tables.

5.5 Effect of Transfer Learning

We now delve into the effects of transfer-learning in DDUp. How much DDUp's transfer-learning via knowledge distillation contributes to better accuracy? We perform experiments where we remove the transfer-learning term of Eq 5. Therefore, we combine the sample from previous data known as the transfer-set with the new update batch and create a model with the same configurations as the base model. Figure 9 shows the results. The results assert that the performance of DDUp is not only related to the previous data sample, and in fact, distillation has a big effect on the improvement of the new models.

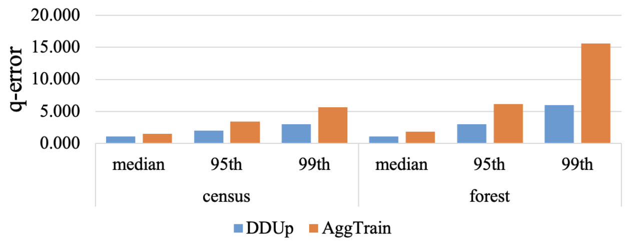


Figure 9: Effect of transfer-learning on q-error. AggTrain, is the case where we aggregate the transfer-set with the new data and train a model similar to the base model.

5.6 Overheads

We report on the costs of each DDUp module separately. All the codes are written and executed in Python 3.8, on an Ubuntu 20 machine with 40 CPU cores, two Nvidia GTX 2080 GPUs and 64GB memory. With respect to memory usage, DDUp performs regular feed-forward steps as in regular NN training. Therefore, DDUp does not increase memory footprints In terms of time, DDUp has two computation costs namely, *OOD detection* and *model update*. OOD detection is split into offline and online phases. Table 11 shows these two times. The largest detection time is for the forest dataset on a Naru model which takes around 3 minutes. However, please note that in the online phase only takes 1 second to detect a change in data.

Table 11: online and offline times during OOD detection.

Dataset	DBEst++		Naru		TVAE	
	off	on	off	on	off	on
census	2.44	0.02	111	1.8	310	5.5
forest	28	0.04	174	0.92	433	8.8
DMV	86	2	144	10	99	0.44

Table 10 shows DDUp's speed up over `retrain` for OOD data, for different update sizes. When data is OOD, DDUp can be over 9× faster than `retrain`. Obviously, speedups will be higher for incremental steps. This fact is reflected in IMDB and TPCH datasets where after inserting the last partition DDUp is 18× faster than `retrain`. Note that the updating time is dependent on a few parameters including update size, transfer-set size, training batch size etc. During updates, we have used smaller training batch sizes. If one tunes the model for bigger batches, and smaller transfer-set sizes, the speed up would be higher.

5.7 Non neural network models

For the sake of completeness and as an additional reference point, we include results for updating a state-of-the-art non-NN model that natively supports data insertions, (DeepDB [21]) used for CE. When an update happens, DeepDB traverses its sum-product-network graph and updates the weights of the intermediate nodes and the histograms at the leaves. We have repeated the same experiment in Table 5 for DeepDB. The results are reported in Table 12.

From Table 12 it can be observed that DeepDB's updating policy is under-performing, as was independently verified in [70]. DDUp (coupled in this experiment with Naru/NeuroCard for CE) always performs better. Nonetheless, we wish to emphasize that the saving grace for DeepDB based on our experiments is that retraining from scratch is very efficient – significantly faster compared to NNs.

6 RELATED WORK

6.1 Learned Database Systems

NN-based components to be used by DBs are emerging rapidly. Different works exploit different neural network models. [17, 77, 78] used generative neural networks to build learned selectivity estimators. Thirumuruganathan et al. [65] used VAEs for AQP. Ma et al. [42] used mixture density networks for AQP. Database indexing research recently has adopted neural networks to approximate cumulative density functions [9, 10, 30, 49]. Query optimization and join ordering are also benefiting from neural networks [27, 45]. Other applications include auto-tuning databases [34, 68, 81], cost estimation [63, 83], and workload forecasting [85].

Among these, this work provides a solution for handling NN model maintenance in the face of insertion-updates with OOD data, when the models need to continue ensuring high accuracy on new and old data and on tasks for which models were originally trained (such as AQP, CE, DG, etc.). While there has been related research on transfer learning for learned DBs such as [20, 74] these target a

Table 12: Performance of DeepDB updating vs. DDUp for Naru, for a CE task in terms of q-error.

Dataset	metric	DeepDB			Naru	
		M_0	update	retrain	M_0	DDUp
census	median	1.05	1.2	1.05	1.08	1.09
	95th	3	4.18	3	2	2
	99th	5.11	8	5	3	3
forest	median	1.02	1.2	1.02	1.04	1.07
	95th	7.5	10.5	7	2.48	3
	99th	31	52	31	4	6
DMV	median	1.06	1.25	1.1	1.02	1.04
	95th	2.5	3.5	2.5	1.20	1.41
	99th	22	37	21	1.83	2.31

different problem setting: They study how to transfer knowledge from a model trained for one task, and/or a DB, and/or a system, and/or a workload to a new task and/or DB, and/or system, and/or workload. They do not study how to keep performing the original task(s) on evolving datasets with insertions carrying OOD data with high accuracy for queries on both old and new data. Simply using these methods by fine-tuning on new data will incur catastrophic forgetting. Nevertheless, since these models employ some sorts of knowledge transfer, they might be useful to support updates. However, it remains open whether and how the models in [20, 74] can be utilized to solve efficiently the problems tackled in this paper. While some of non-neural-network models (e.g., DeepDB) can very efficiently retrain from scratch, NN-based models for the above problem setting either do not support insertion-updates or suffer from poor accuracy when facing OOD data, unless paying the high costs of retraining from scratch.

6.2 OOD Detection

OOD detection has recently attracted a lot of attention and it has long been studied in statistics as concept drift (CD) detection, or novelty detection. In general, CD and OOD detection methods could be divided into two broad categories [14, 39, 69]: First, prediction-based methods, which use the predictions of the underlying models to test for a change. Recent ML models usually use the predictive probabilities of the classifiers as a confidence score to identify changes [23, 53, 58, 72]. Others may monitor the error of the underlying models and trigger an OOD signal when a significant change is captured [2, 13, 37, 50, 60]. While these approaches are very efficient in time, they typically come with limiting assumptions depending on the underlying model or application. For example, most of them can only be utilized and are only studied for classification (supervised) tasks. The second broad family of methods is distribution-based methods. Some of these methods try to find a distance measure that can best show the discrepancy between new data and old data distributions, using tests like Kolmogorov-Smirnov (KS), [29], Wilcoxon [55], and their multi-variate variants [3, 11]. Others try to learn the density of the underlying data distribution test for a significant change, like kernel-density-based approaches [4, 8, 16, 25, 40, 64]. More recent works utilize the estimated likelihoods of generative models [47, 57, 75]. Other approaches rely on the inner representations of the networks [19, 32, 33]. Nonetheless, this second family of OOD detection methods are usually expensive (esp. for multi-dimensional data) and involve fitting a separate density estimator. Hence, the main problem is that in an insertion scenario, the density estimators also need to be updated (typically via training from scratch, upon each insertion).

6.3 Incremental Learning (IL)

Most IL methods regularize the model in a way that it acquires knowledge from the new task while retaining the knowledge of old tasks. For example, *Elastic Weight Consolidation (EWC)* [28] adds a regularizer to control the learning speed around important weights of the network for old tasks while learning a new task. Similar works are developed around this idea [31, 36, 67], *Path Integral (PathInt)* [80] ,*Riemanian Walk (RWalk)* [6]. Other approaches exploit knowledge distillation to retain the knowledge of previous tasks [35]. Another group of IL methods, save exemplars from past data [5, 56, 73] or generate samples/features using generative models [24, 52] and involve them in learning new tasks. Lopez et al. [38] has proposed *Gradient Episodic Memory* that consists of M blocks of memory to store examples from T tasks and uses the model's prediction on these examples as a constraining loss that inhibits the model to bias toward new task and forget past tasks. Lastly, some works try to completely keep previous models and create new models (or part of a model like a single layer) for each new task. Aljundi et al. [1] introduce *Expert Gate* with different models for each task and an autoencoder which learns the representations of each task to assign test-time tasks to the proper model. Instead of learning a whole new model, Rusu et al. [59] introduce *Progressing Neural Networks* which add new columns to the previous network architecture and learns lateral connections between them. Most of the above methods, do not account for in- and out- of distribution updates and are not easily extendable to different learning tasks.

7 CONCLUSION

Learned DB components can become highly inaccurate when faced with new OOD data when aiming to ensure high accuracy for queries on old and new data for their original learning tasks. This work proposes, to our knowledge, the first solution to this problem, coined DDUp. DDUp entails two novel components, for OOD detection and model updating. To make detection widely applicable, OOD detection in DDUp exploits the output of the neural network (be it based on log-likelihood, cross-entropy, ELBO loss, etc.), and utilizes a principled two-sample test and a bootstrapping method to efficiently derive and use thresholds to signal OOD data. DDUp also offers a general solution for model updating based on sequential self-distillation and a new loss function which carefully accounts for *catastrophic forgetting* and *intransigence.* This work showcases the wide applicability of DDUp model updating by instantiating the general approach to three important learned functions for data management, namely AQP, CE, and DG, whereby a different type of NN (MDNs, DARNs, VAEs) is used for each. In fact, to our knowledge, no prior work has shown how to "distill-and-update" MDNs, VAEs, and DARNs. Comprehensive experimentation showcases that DDUp detects OOD accurately and ensures high accuracy with its updated models with very low overheads.

8 ACKNOWLEDGEMENT

This work is partially sponsored by Huawei IRC and by EPSRC while doing a PhD at the University of Warwick.

REFERENCES

[1] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. 2017. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3366–3375.

[2] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and Rafael Morales-Bueno. 2006. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, Vol. 6. 77–86.

[3] Ludwig Baringhaus and Carsten Franz. 2004. On a new multivariate two-sample test. *Journal of multivariate analysis* 88, 1 (2004), 190–206.

[4] Li Bu, Cesare Alippi, and Dongbin Zhao. 2016. A pdf-free change detection test based on density difference estimation. *IEEE transactions on neural networks and learning systems* 29, 2 (2016), 324–334.

[5] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*. 233–248.

[6] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 532–547.

[7] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. 2017. Generating multi-label discrete patient records using generative adversarial networks. In *Machine learning for healthcare conference*. PMLR, 286–305.

[8] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. 2006. An information-theoretic approach to detecting changes in multidimensional data streams. In *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*. Citeseer.

[9] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. 2020. ALEX: an updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 969–984.

[10] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *arXiv preprint arXiv:2006.13282* (2020).

[11] Giovanni Fasano and Alberto Franceschini. 1987. A multidimensional version of the Kolmogorov–Smirnov test. *Monthly Notices of the Royal Astronomical Society* 225, 1 (1987), 155–170.

[12] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born again neural networks. In *International Conference on Machine Learning*. PMLR, 1607–1616.

[13] Joao Gama and Gladys Castillo. 2006. Learning with local drift detection. In *International conference on advanced data mining and applications*. Springer, 42–55.

[14] 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.

[15] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9304–9312.

[16] Feng Gu, Guangquan Zhang, Jie Lu, and Chin-Teng Lin. 2016. Concept drift detection based on equal density estimation. In *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 24–30.

[17] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep learning models for selectivity estimation of multiattribute queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1035–1050.

[18] Dan Hendrycks and Kevin Gimpel. 2017. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks.. In *ICLR*.

[19] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. 2019. Using pre-training can improve model robustness and uncertainty. In *International Conference on Machine Learning*. PMLR, 2712–2721.

[20] Benjamin Hilprecht and Carsten Binnig. 2021. One Model to Rule them All: Towards Zero-Shot Learning for Databases. *arXiv preprint arXiv:2105.00642* (2021).

[21] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2019. Deepdb: Learn from data, not from queries! *arXiv preprint arXiv:1909.00607* (2019).

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[23] Heinrich Jiang, Been Kim, Melody Guan, and Maya Gupta. 2018. To trust or not to trust a classifier. *Advances in neural information processing systems* 31 (2018).

[24] Ronald Kemker and Christopher Kanan. 2017. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563* (2017).

[25] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. 2004. Detecting change in data streams. In *VLDB*, Vol. 4. Toronto, Canada, 180–191.

[26] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

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

[28] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

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

[30] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*. 489–504.

[31] Janghyeon Lee, Hyeong Gwon Hong, Donggyu Joo, and Junmo Kim. 2020. Continual learning with extended kronecker-factored approximate curvature. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9001–9010.

[32] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems* 31 (2018).

[33] Chun-Liang Li, Kihyuk Sohn, Jinsung Yoon, and Tomas Pfister. 2021. Cutpaste: Self-supervised learning for anomaly detection and localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9664–9674.

[34] 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.

[35] Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence* 40, 12 (2017), 2935–2947.

[36] Xialei Liu, Marc Masana, Luis Herranz, Joost Van de Weijer, Antonio M Lopez, and Andrew D Bagdanov. 2018. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2262–2268.

[37] David Lopez-Paz and Maxime Oquab. 2016. Revisiting classifier two-sample tests. *arXiv preprint arXiv:1610.06545* (2016).

[38] David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems* 30 (2017), 6467–6476.

[39] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2018), 2346–2363.

[40] Ning Lu, Guangquan Zhang, and Jie Lu. 2014. Concept drift detection via competence models. *Artificial Intelligence* 209 (2014), 11–28.

[41] 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*. 175–191.

[42] Qingzhi Ma, Ali Mohammadi Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, and Peter Triantafillou. 2021. Learned Approximate Query Processing: Make it Light, Accurate and Fast.. In *CIDR*.

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

[44] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*. 1275–1288.

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

[46] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[47] Warren Morningstar, Cusuh Ham, Andrew Gallagher, Balaji Lakshminarayanan, Alex Alemi, and Joshua Dillon. 2021. Density of states estimation for out of distribution detection. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 3232–3240.

[48] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. 2018. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136* (2018).

[49] 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.

[50] Rimma V Nehme, Elke A Rundensteiner, and Elisa Bertino. 2009. Self-tuning query mesh for adaptive multi-route query processing. In *Proceedings of the 12th*

International Conference on Extending Database Technology: Advances in Database Technology. 803–814.

[51] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 427–436.

[52] Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. 2019. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 11321–11329.

[53] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. 2019. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems* 32 (2019).

[54] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. 2018. Data synthesis based on generative adversarial networks. *arXiv preprint arXiv:1806.03384* (2018).

[55] Francisco Pereira, Tom Mitchell, and Matthew Botvinick. 2009. Machine learning classifiers and fMRI: a tutorial overview. *Neuroimage* 45, 1 (2009), S199–S209.

[56] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition.* 2001–2010.

[57] Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. 2019. Likelihood ratios for out-of-distribution detection. *Advances in Neural Information Processing Systems* 32 (2019).

[58] Lukas Ruff, Jacob R Kauffmann, Robert A Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G Dietterich, and Klaus-Robert Müller. 2021. A unifying review of deep and shallow anomaly detection. *Proc. IEEE* (2021).

[59] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).

[60] Fotis Savva, Christos Anagnostopoulos, and Peter Triantafillou. 2019. Aggregate query prediction under dynamic workloads. In *2019 IEEE International Conference on Big Data (Big Data).* IEEE, 671–676.

[61] Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. 2021. Remember what you want to forget: Algorithms for machine unlearning. *Advances in Neural Information Processing Systems* 34 (2021), 18075–18086.

[62] Ali Mohammadi Shanghooshabad, Meghdad Kurmanji, Qingzhi Ma, Michael Shekelyan, Mehrdad Almasi, and Peter Triantafillou. 2021. PGMJoins: Random Join Sampling with Graphical Models. In *Proceedings of the 2021 International Conference on Management of Data.* 1610–1622.

[63] 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.

[64] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. 2007. Statistical change detection for multi-dimensional data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining.* 667–676.

[65] Saravanan Thirumuruganathan, Shohedul Hasan, Nick Koudas, and Gautam Das. 2020. Approximate query processing for data exploration using deep generative models. In *2020 IEEE 36th International Conference on Data Engineering (ICDE).* IEEE, 1309–1320.

[66] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B. Tenenbaum, and Phillip Isola. 2020. Rethinking Few-Shot Image Classification: a Good Embedding Is All You Need?. In *https://arxiv.org/abs/2003.11539.*

[67] Michalis K Titsias, Jonathan Schwarz, Alexander G de G Matthews, Razvan Pascanu, and Yee Whye Teh. 2019. Functional regularisation for continual learning with gaussian processes. *arXiv preprint arXiv:1901.11356* (2019).

[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] K Wang, Paul Vicol, Eleni Triantafillou, and Richard Zemel. 2020. Few-shot Out-of-Distribution Detection. In *ICML Workshop on Uncertainty and Robustness in Deep Learning.*

[70] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2020. Are We Ready For Learned Cardinality Estimation? *arXiv preprint arXiv:2012.06743* (2020).

[71] AJ Watkins and KV Mardia. 1992. Maximum likelihood estimation and prediction mean square error in the spatial linear model. *Journal of Applied Statistics* 19, 1 (1992), 49–59.

[72] Andrew G Wilson and Pavel Izmailov. 2020. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems* 33 (2020), 4697–4708.

[73] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. 2019. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 374–382.

[74] Ziniu Wu, Peilun Yang, Pei Yu, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2021. A unified transferable model for ml-enhanced dbms. *arXiv preprint arXiv:2105.02418* (2021).

[75] Zhisheng Xiao, Qing Yan, and Yali Amit. 2020. Likelihood regret: An out-of-distribution detection score for variational auto-encoder. *Advances in neural information processing systems* 33 (2020), 20685–20696.

[76] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling tabular data using conditional gan. *Advances in Neural Information Processing Systems* 32 (2019).

[77] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: one cardinality estimator for all tables. *arXiv preprint arXiv:2006.08109* (2020).

[78] 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. *arXiv preprint arXiv:1905.04278* (2019).

[79] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 4133–4141.

[80] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence. In *International Conference on Machine Learning.* PMLR, 3987–3995.

[81] 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.

[82] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random sampling over joins revisited. In *Proceedings of the 2018 International Conference on Management of Data.* 1525–1539.

[83] 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.

[84] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2020. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *arXiv preprint arXiv:2011.09022* (2020).

[85] Yonghua Zhu, Weilin Zhang, Yihai Chen, and Honghao Gao. 2019. A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment. *EURASIP Journal on Wireless Communications and Networking* 2019, 1 (2019), 1–18.