Active Learning for ML Enhanced Database Systems

Lin Ma*
Carnegie Mellon University
lin.ma@cs.cmu.edu

Bailu Ding
Microsoft Research
badin@microsoft.com

Sudipto Das*
Amazon Web Services
sudiptdas@gmail.com

Adith Swaminathan
Microsoft Research
adswamin@microsoft.com

ABSTRACT

Recent research has shown promising results by using machine learning (ML) techniques to improve the performance of database systems, e.g., in query optimization or index recommendation. However, in many production deployments, the ML models' performance degrades significantly when the test data diverges from the data used to train these models.

In this paper, we address this performance degradation by using B-instances to collect additional data during deployment. We propose an active data collection platform, ADCP, that employs active learning (AL) to gather relevant data cost-effectively. We develop a novel AL technique, Holistic Active Learner (HAL), that robustly combines multiple noisy signals for data gathering in the context of database applications. HAL applies to various ML tasks, budget sizes, cost types, and budgeting interfaces for database applications. We evaluate ADCP on both industry-standard benchmarks and real customer workloads. Our evaluation shows that, compared with other baselines, our technique improves ML models' prediction performance by up to 2× with the same cost budget. In particular, on production workloads, our technique reduces the prediction error of ML models by 75% using about 100 additionally collected queries.

ACM Reference Format:
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 (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3318464.3389768

*Lin Ma and Sudipto Das performed the work while at Microsoft Research.

1 INTRODUCTION

We increasingly see the promise of using machine learning (ML) techniques to enhance database systems' performance, such as in query run-time prediction [18, 37], configuration tuning [51, 66, 77], query optimization [35, 44, 50], and index tuning [5, 14, 61]. For example, a query optimizer can use an ML model that predicts a query plan's execution cost to search for the best query plan [45]. Most of these techniques use supervised learning [54], which trains the ML model using a *labeled dataset.*

Supervised learning performs well when the labeled data used for training the ML model comes from the same distribution as the data that the model will be deployed to make predictions for [17, 20]. In many ML applications for databases, however, the data distribution seen after deployment differs from that of the training data due to the databases' complexity and their diverse workloads. For example, the input of a model that predicts execution cost is often based on plan information, including operator types and join orders, which heavily depends on the content of the database and the queries [14, 18, 37, 44]. The workloads in the production environment where the model is deployed, however, may come from different databases or queries than the workloads that are used to train the ML model. This results in substantially different inputs fed to the deployed ML model than what it saw during training.

ML models can make huge prediction errors when they are used for predicting on the data that differs from what they are trained with [14, 37]. We demonstrate this phenomenon in detail in Section 2.2. Fundamentally, the training data's quality and coverage of the production workload determines the model's prediction performance. However, we do not know what the production workload will be until we actually deploy a model. Thus, we can not assess a priori whether we have good enough training data nor whether ML models will perform well for our workloads.

Conceptually, if we had a training dataset that covered *all* possible workloads, we could guarantee that ML models trained on them will perform well when deployed. Unfortunately, it is infeasible to enumerable all possible databases and workloads to collect such a massive dataset.

Instead, we aim at quickly adapting a model to the current workload after the model is deployed, by collecting additional execution data that is most beneficial to improve the model's performance within a given cost budget. We then retrain the ML model with this additional data so that it adapts to the observed workload. There are three key questions with this approach: What mechanism do we use to collect labeled data in production? How can we gather labeled data that is relevant to the ML model? And how can we make such data collection scalable for the increasing number of ML applications for databases?

To gather labeled data from production workloads, we leverage fork-offs (i.e., B-instances [13]) for databases in the cloud or backups for on-premise databases. These replicas can execute and instrument queries from production workloads to collect telemetry for data labeling.

To gather labeled data relevant to ML models, **our key insight** is that, based on the database application and workloads, we can narrow down the space of data that the ML models need to predict for. In many database applications, we observe that we can derive the data space that the ML model needs to predict for based on how the application uses the model. For example, the optimizer uses a plan cost prediction model to navigate its search space, where the search space for a set of queries, i.e., the potential pool of test data, can be recorded. By executing additional plans from this search space and recording their execution costs, we can improve the model's prediction specifically for this pool of test data. Such a *workload-centric* data collection can focus on gathering the most helpful labels for the ML model.

Finally, there are many workloads during the lifetime of an ML model and various tasks in ML-enhanced databases – how can we scalably collect data for all of them? We present an *active data collection platform* (ADCP) that manages the complexity of this data collection (see Section 3). The ADCP takes a base ML model trained with some training data and a budget specified either by resource cost or the number of labels to collect from the production workload. The ADCP then uses additional B-instances or replicas to collect a subset of labeled data under the budget, and it returns the new labels for model retraining. The main challenge for such a platform is how to intelligently decide which labels to collect to best improve the model prediction on a specific set of data.

We formulate this problem as pool-based *active learning* (AL) [57], where an AL strategy selects the best training data from a pool of unlabeled points. Although AL is known to be superior to rudimentary approaches (e.g., randomly selecting points) in many domains [56], ADCP faces a set of holistic challenges neglected by prior work in AL (see Section 8):

❶ **Robustness**: AL strategies rely on several signals to decide which data points to label. The signals can come from the ML model, the unlabeled data distribution, or the labeling cost estimates. Though these signals can be available in our context (see Section 3.2), they are often noisy and unreliable.
❷ **Cost Sensitivity**: Typical AL strategies assume uniform labeling costs. But the resources needed to acquire labels in databases can be drastically different, e.g., the disparate plan execution costs with different selectivities or join orders.
❸ **Batching**: Many AL strategies assume model retraining after acquiring each label. Training ML models for databases can be expensive [14, 45]. Thus, the ADCP needs to support acquiring a batch of labels before retraining the model.

We present a simple and effective AL strategy, *Holistic Active Learner (HAL)*, that addresses all these challenges (see Section 4). HAL takes the labeling cost into account and robustly combines multiple noisy signals. We discuss HAL's design choices in Section 5 and explain HAL's execution in Section 6. We evaluate HAL with industry-standard benchmarks and real-world workloads for a variety of ML applications in databases, including different ML tasks, ML algorithms, cost types, budget sizes, and budgeting interfaces (see Section 7). Our results show that the ADCP with HAL consistently outperforms the state-of-the-art AL strategies for improving the prediction accuracy on the target test data. In particular, our ADCP can reduce ML model's prediction error on production workloads by 75% with ∼100 additionally executed queries.

2 BACKGROUND AND MOTIVATION

We first discuss the typical ML tasks encountered in ML enhanced databases and how they have prediction errors in production. We then discuss the opportunity to use additional resources with active learning techniques to collect extra labels to adapt these models.

2.1 ML Enhanced Databases

There are many examples of ML enhanced databases [14, 37, 45]. Without loss of generality, we discuss two examples that represent two distinct types of ML problems which have many applications in database systems. We use these two tasks throughout the paper, as well as in the evaluation.

Execution cost prediction (**ECP**) is a *regression* ML task that takes in a query plan and outputs a real number that represents the estimated CPU time to execute the plan. ECP has several applications in databases. For example, cost-based query optimizers enumerate candidate plans for a query and select the plan with the lowest estimated cost to execute [10]. Advanced index tuners also rely on the plan's estimated execution cost under different index configurations to recommend indexes with the highest cost reduction [2, 11, 12, 78]. ECP can help the optimizer/tuner's search by predicting the candidate plans' costs. We can also extend this task formulation to predict other logical resources, e.g., logical I/O.

Plan regression prediction (**PRP**) is a *classification* ML task that takes in a pair of plans and outputs a class nominal

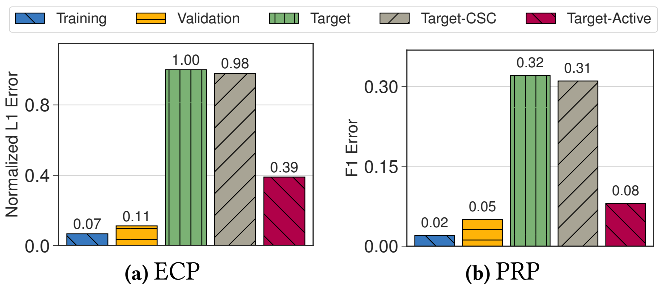


Figure 1: Target Data Prediction Error – ML model's predictions error increase with training-target data mismatch. Active data collection reduces this error.

that represents whether the first plan has a higher execution cost than the second plan. Various database applications can leverage PRP for improvement. For example, PRP can help the optimizer to compare the costs of candidate plans or identify the query plan regressions with changed plans [15]. It may also significantly help an index tuner to detect performance regressions on candidate index configurations [14], which is a requirement for state-of-the-art index tuners [13].

Previous works have applied various ML models to these tasks, such as random forests [15], boosted regression trees [37], and deep neural networks [45]. Training data for these models typically come from the execution history of standard benchmarks or any accessible databases. For example, both ECP and PRP can use the recorded query plan execution costs in the workloads as training data. Most previous works on ML enhanced databases assume that these models' test data are similar to the training data. Prior work has also shown that the prediction error can increase significantly when the models are deployed for different data distribution [14].

2.2 Prediction Error in Production

The prediction error of the ML model can be dramatically higher in a production environment compared with evaluation on held-out training data. As an example, we simulate the production deployment of an ML enhanced index tuner. We use a state-of-the-art index tuner [13] to generate the plan (plan pair) space on 14 diverse database workloads (details in Section 7.2). We simulate a new deployment by holding-out the data in one database (i.e., the *target* database) and using the labeled data in the remaining 13 databases to train ML models for both ECP and PRP, resulting in 14 simulation runs per model in total. We also split the labeled training data into a 80% training set and a 20% validation set. We investigate a number of advanced ML techniques and use the random forest that has the best prediction. The trained models then predict the labels for the data points generated from the held-out database (target data). For ECP, we measure the normalized L_1 error ($|Prediction - Actual|$) on the target data. For PRP, we measure the prediction's F1 error [14]. Both errors are between 0 (best) and 1 (worst).

Figure 1 shows the average model performance across 14 databases. For both tasks, the model achieves high prediction performance on both the training and the validation data (denoted as **Training** and **Validation**). This indicates good generalization ability of the ML model and no-overfitting [20]. However, the target predictions (denoted as **Target**) have huge errors (9× higher L_1 error or 6× higher F1 error compared to validation data prediction), which is caused by data mismatch between the training and the target data [65].

We also evaluate popular covariate shift correction (CSC) techniques to address the distribution mismatch between training and target [1, 6, 26]. When the training data overlaps with the target data but only differs in the distribution, CSC can adjust the training weight of the labeled data to match the target data distribution. However, we observe that the training data and the target can contain disparate regions in the input space in our database workloads, where CSC cannot help [65]. As shown in Figure 1, the best performing CSC technique (denoted as **Target-CSC**) does not improve prediction quality substantially.

Thus, neither changing the supervised ML algorithm nor reweighting the training data can easily address the ML models' degradation in production.

2.3 Workload-Centric Data Collection with B-Instances / Replicas

We propose to collect extra labels from an unlabeled dataset to reduce the ML model's generalization error during production. For example, a query optimizer explores many possible plans in the plan space for queries in the production workload. This plan space produces a pool of unlabeled datapoints. The database only executes the best-estimated plan, so the labels (e.g., execution costs) for alternative plans are not available via normal operation. The ML model for this specific workload can benefit from collecting additional labels from the plan space, i.e., the target data. In ML literature, predicting labels for a given and unlabeled test (target) data is called the *transductive* setting [29, 76]. Transductive data collection can leverage the information from the unlabeled target data to acquire more valuable labels. This differs from the *inductive* setting more commonly used in academia, where the test data is only used during evaluation. Evaluating ML models in the transductive setting needs to be careful and we describe the protocol for making fair comparisons between transductive algorithms in Section 7.1.

To derive labels for any unlabeled data-point, an ML enhanced database can extract query execution statistics. For example, labels for ECP and PRP are the execution costs (or cost differences) of query plans. We can use the B-instance

/ replica of production databases to collect such data for alternative plans that were not executed in production.

In Figure 1, we also highlight that by intelligently executing ~100 queries on the B-instance, our proposed technique (denoted as **Target-Active**) reduces the target data prediction error for ECP by 61% and for PRP by 75%.

2.4 Active Learning

The target data, e.g., the optimizer/index tuner's search space, may contain tens of thousands of plans that can consume a lot of resources to execute exhaustively on B-instances. We leverage active learning (AL) techniques to select the most valuable plans and reduce this labeling cost. We first describe the typical AL setup [57]:

An AL strategy takes as input a set of labeled training data X_L, a set of unlabeled target data X_U, and a base ML model θ. X_L and X_U are assumed to come from the same underlying data distribution. A loss function $L_{X_U}(\theta^{X_L}(x), y(x))$ denotes the loss of the model θ trained with data X_L, compared to the true label function $y(x)$, and evaluated on X_U. A typical AL strategy derives an *informativeness* signal for each data point $x \in X_U$ using the model θ to estimate how much x's label can reduce the loss L_{X_U}. We refer to this signal as weight w_x. For instance, w_x can be an estimate of the model's confidence or uncertainty in its prediction for x. The AL strategy selects the data point with the highest w_x to label and retrains the ML model – when w_x is uncertainty, this yields the *uncertainty sampling* class of AL algorithms [36]. In the next round, the AL uses updated estimates w_x from the retrained θ to select the next data-point. This proceeds until either the loss of θ is small enough or until the labeling budget is exhausted.

We will discuss the additional AL challenges in the database context in Section 3.3 and our re-formulation of the AL problem in Section 4.1.

3 ACTIVE DATA COLLECTION PLATFORM

We propose an active data collection platform (ADCP) to collect labels to adapt the ML model to the production workload. We now discuss the ADCP's architecture and workflow. We then discuss the challenges in applying AL for deciding which target data-points to label.

3.1 Platform Architecture and Workflow

The ADCP uses additional resources to collect the labels during production workloads. As shown in Figure 2, an ADCP connects to the ML applications in the production database and the labeling handlers in the B-instances / replicas. The ADCP can acquire the target test data and the ML model from the specific ML application in the database, e.g., the optimizer's plan search space for the production workload

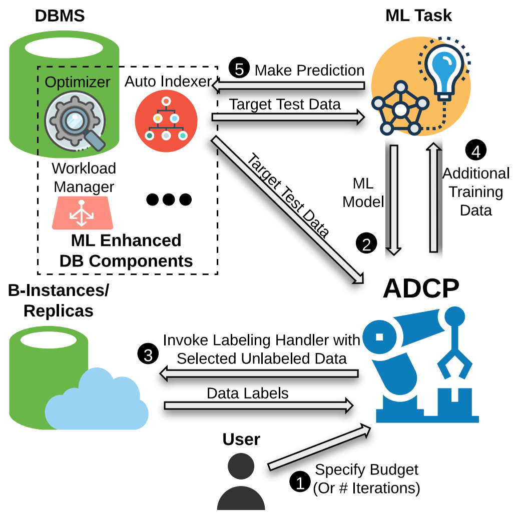


Figure 2: Active Data Collection Platform (ADCP) – The ADCP actively collects training data with additional resources and returns new labels to retrain the ML model.

and the ECP (PRP) model used by the optimizer. The ADCP can also invoke the labeling handler to execute a query plan and acquire its label, e.g., the plan's execution cost, for model retraining.

We anticipate that users will adopt the ADCP in the following steps. ❶ The users specify the ML application whose model needs improvement and the resource budget for collecting data. ❷ The ADCP acquires the target (unlabeled) data generated by the production database and the ML model's predictions on the target data. ❸ The ADCP selects a batch of points from the target data under the resource budget, and invokes labeling handlers to get the actual labels for these points. ❹ The ML model is retrained with the newly collected data. ❺ The new model is installed back to the ML application to make predictions for the ML enhanced components in the production database.

The users may optionally specify an iteration number for the data collection and model retraining before installing the model back to the application. In this case, the ADCP will repeat ❸ and ❹ for the specified number of iterations while ensuring that the resources consumed stay under budget.

3.2 Application and Usage Scenarios

In Section 2.3 we outlined how the pool of unlabeled data points (target data) is created in the ADCP (see Figure 2). Given this target data, ADCP supports several scenarios to allow a user to specify how to collect additional data.

- **Budget:** Since users are often cost-sensitive, ADCP allows the user to specify a budget for labeling. This can be a fixed number, say n data points (typical in AL) or richer notions like "spend at most C cost", per iteration. We refer to such a choice as the *budgeting interface*. ADCP provides this flexibility since different data points can take dramatically different costs to label.

- **Cost:** ADCP allows the user to provide a labeling cost estimator, and selects data-points in a cost-sensitive way. In our example applications (Section 2.1), the costs can contain the query execution cost and the index creation cost. The true labeling cost is typically unavailable or can even be the prediction goal of the ML model. But, we can leverage crude estimators in databases, such as the optimizer's estimates. Such a crude estimator can be better than the ML models trained with mismatched training data, but can also be inaccurate and noisy (shown in Section 7). Thus, we design the ADCP to leverage imperfect cost information robustly, and jointly optimizing the cost estimator with the data collection is interesting future work. If a cost estimator is unavailable, the ADCP can also fall back to the plan budgeting interface with uniform labeling costs.
- **Uncertainty:** Users provide a base ML model to ADCP, which handles the training of the ML model and then queries the model to produce informativeness scores w_x that can drive the data-collection policy.
- **Retraining frequency:** In ML enhanced databases, retraining the ML model can be very expensive. The ADCP supports specifying how many times this retraining should happen by allowing the user to set the number of iterations of data collection. ADCP does not yet include this model retraining cost into the user-specified budgets or costs, and we believe that estimating/reasoning about these retraining costs is an interesting avenue for future work.

3.3 Active Learning Challenges

There are three major challenges to apply AL for the ADCP to decide which points to label:

Robustness: AL strategies typically select points based on their informativeness w_x derived from the ML model (Section 2.4), since not all points may benefit the model equally. For example, query plans that are less similar to the model's training data may improve the model's prediction much more than other plans. For ADCP, the ML model's informativeness signal can be unreliable because of the data distribution mismatch faced by the model θ. As discussed in Section 3.2, other signals that the AL strategy may leverage, e.g., the cost estimator, can also be noisy and unreliable.

Cost-Sensitive: Labeling each point comes with a cost, e.g., executing a query plan. Such cost can be dramatically different from one data point to another because of the plans' diverse operators, cardinalities, or join orders. The index cost type further complicates this problem because query plans can share the same index and have interdependent costs.

Notation	Description
X_L	the set of labeled (training) data points
X_U	the set of unlabeled (target) data points
X_A	the set of data points selected by the AL strategy to label
$y(x)$	the true label of a data point x
θ	an ML model
$c(x)$	the cost to label a data point x
$u(x)$	the uncertainty of a data point x
w_x	the weight of a data point x used by AL strategies for selection
B	the resource budget for the data collection
γ	the redundancy rejection threshold based on cluster size
β	the redundancy rejection threshold based on cluster uncertainty

Table 1: Table of Important Notations

Batching: Many ML tasks for databases use models that are expensive to train [14, 45]. Thus, the ADCP requires the AL strategy to label a batch of points at once before model retraining to reduce the training overhead. An AL strategy that only focuses on points with higher informativeness or lower costs may select a number of points in a batch with similar information, and hence select sub-optimal batches.

Since we envision the ADCP to support a wide variety of ML applications for databases, we also design the AL strategy to make minimal assumptions about the ML task or model.

We next describe our formulation of active learning that augments the typical setting (Section 2.4) and a novel AL strategy that surmounts the challenges described above.

4 HOLISTIC ACTIVE LEARNER

We present a novel AL strategy, *Holistic Active Learner* (**HAL**), that addresses the ADCP's AL challenges discussed in Section 3.3. We first explain HAL's main concepts and core framework, which are simple yet effective. We defer discussion of HAL's design space and our decisions to Section 5.

4.1 Notation and Formulation

An AL strategy in our problem setting takes as input a set of labeled training data X_L, a set of unlabeled target data X_U, a base ML model θ, a labeling budget B, and the cost $c(x)$ to acquire the label for a data point x. The core addition to the vanilla AL formulation is the notion of labeling budget B and labeling costs $c(x)$. Table 1 summarizes the important notations used in the paper.

The AL strategy selects a batch of points X_A from the target data X_U under the budget B, as:

$$\arg\min_{X_A} \; L_{X_U}(\theta^{X_L \cup X_A}(x), y(x))$$

subject to

$$X_A \subseteq X_U, \sum_{x \in X_A} c(x) \leq B$$

4.2 Biased Sampling

We first highlight how HAL addresses the robustness challenge. A deterministic optimization strategy to select points with the highest w_x, which is typical in the AL literature [36,

Algorithm 1: Holistic Active Learner

```
Input  : $X_L$ is the original labeled training data,
         $X_U$ is the unlabeled target data,
         $\theta$ is an ML model that predicts the data label,
         $B$ is the budget for data collection,
         $c$ is a function to estimate the labeling cost,
         $\mathcal{P}$ is a function to calculate the sampling weight,
         $\mathcal{R}$ is a function to identify the redundant points
Output : $X_A$ is the batch of selected points to label, where
         $X_A \subseteq X_U$ and $\sum_{x \in X_A} c(x) \leq B$
1  $W \leftarrow \mathcal{P}(X_U, \theta, c)$  // $w_x \in W$ is $x \in X_U$'s sampling weight
2  $X_A \leftarrow \phi$  // the points selected by the AL strategy
3  $X_Q \leftarrow \{x | c(x) \leq B, x \in X_U\}$  // the qualified points for the AL strategy to select from
4  $X_R \leftarrow \phi$  // the redundant points that the strategy should reject
5  while $X_Q \setminus X_R \neq \phi$ do
6      Sample $\tilde{x}$ from $X_Q \setminus X_R$ according to weight $W$
7      $X_A = X_A \cup \{\tilde{x}\}$  // add $x$ to the selected points
8      $X_R = \mathcal{R}(X_A, X_U, W)$  // update the redundant points
9      $B = B - c(\tilde{x})$  // update the budget
10     $X_Q \leftarrow \{x | c(x) \leq B, x \in X_Q \setminus \{\tilde{x}\}\}$  // update the qualified points
11 end
12 return $X_A$
```

57, 62, 73], is doomed to be brittle when there is noise and variance in w_x.

HAL, in contrast, adopts a biased-sampling approach. HAL also defines the informativeness score w_x, but only uses w_x as a sampling weight and employs probabilistic sampling. Thus, HAL prefers points with higher w_x, but does not entirely depend on the quality of w_x. In essence, HAL incorporates randomness into w_x to resist w_x's noise and variance. We provide a deeper analysis on this insight in Section 5.1.

4.3 Cost-weighting

Under the sampling framework, HAL achieves cost-sensitivity by incorporating the labeling cost $c(x)$ into the sampling weight w_x. We denote the function that combines the informativeness and the cost to derive w_x as $\mathcal{P}$, which essentially calculates the "per cost unit" informativeness. We discuss $\mathcal{P}$'s several design choices in Section 5.2.

4.4 Redundancy Rejection

We now highlight how HAL combats the batching challenge. While HAL can easily support batching by sampling without replacement, there can be redundant samples in a batch. HAL uses a clustering-based approach to explicitly capture the target data distribution and identify the redundant points (denoted as $\mathcal{R}$). HAL rejects samples from the well-represented clusters in the batch. We discuss this design and two novel adaptations to derive the rejection scheme in Section 5.3.

4.5 Algorithm Framework

Algorithm 1 shows the algorithm of HAL. Recall that $\mathcal{P}$ returns the sampling weight w_x for a data point x and $\mathcal{R}$ returns the set of redundant points in the target data. And we discuss $\mathcal{P}$ and $\mathcal{R}$ in later sections.

In Line 1, HAL first calculates the sampling weight w_x with $\mathcal{P}$. HAL then initializes the set of selected points X_A as ϕ (the empty set), the set of qualified sampling points X_Q as the points whose labeling cost is under the remaining budget, and the set of redundant points X_R as ϕ (Line 2-4). HAL then repeats a sampling loop until the budget is exhausted (Line 5-10). In each loop, HAL first samples a point $\tilde{x}$ under the sampling weight w_x from the qualified and non-redundant points, i.e., $X_Q \setminus X_R$ (Line 6). It then adds $\tilde{x}$ to the set of selected points X_A (Line 7), and updates X_R with function $\mathcal{R}$ (Line 8). It then calculates the remaining budget and updates qualified points X_Q based on the current budget (Line 9-10).

When the set of qualified points X_Q becomes ϕ, HAL returns X_A and terminates (Line 12).

5 DESIGN SPACE AND DECISIONS

There are many design choices in HAL, for instance, in setting $\mathcal{P}$ and $\mathcal{R}$. We now explore the design space and discuss our decisions in detail.

5.1 Biased-Sampling Decisions

To design an AL strategy for the ADCP, it is natural to investigate the state-of-the-art AL strategies. We initially set out to evaluate a number of AL strategies that might be able to handle the challenges in Section 3.3, ranging from simple strategies that focus on the most informative/cheap points to complex strategies that combine multiple signals together [36, 57, 62, 73]. We find that none of them select points that give satisfactory prediction improvement for the ML models (Section 7.4).

As mentioned in Section 4.2, all these strategies select points deterministically to maximize the total informativeness score w_x. Recent research in AL provides theoretical proof and empirical evidence that probabilistic sampling (e.g., uniform random sampling) outperforms advanced AL strategies in certain cases, such as when the dataset is noisy [48]. Uniform random sampling does not require any additional signal, but is oblivious of the different informativeness and cost values of datapoints, and does not perform well in ADCP's scenarios either (Section 7).

HAL uses w_x as the sampling weight for a biased *softmax* sampling process, which interpolates between deterministically maximizing w_x and uniform random sampling. Concretely, HAL's biased sampling process with weight w_x is defined as

$$p(x) = \frac{w_x}{\sum_{x'} w_{x'}} \tag{1}$$

where $p(x)$ is the sampling probability of x. By the Gumbel trick [21, 42], sampling one point x with probability distribution $p(x)$ is equivalent to independently perturbing the log-probabilities $\log p(x)$ with Gumbel noise and finding the largest element:

$$\arg\max_x \ \log p(x) + G_x \tag{2}$$

where $G_x \sim \mathit{Gumbel}(0, 1)$. Furthermore, recent extensions of the Gumbel trick note that taking the top k largest perturbed log-probabilities (instead of the maximum) yields k samples from the probability distribution $p(x)$ without replacement [33]. This is equivalent to a greedy optimization algorithm that maximizes $\log p(x) + G_x$ for the k samples (note that $\log p(x)$ is a monotone transformation of w_x).

Thus, by only keeping $\log p(x)$ in Equation (2), we recover many deterministic AL strategies that maximize the "informativeness score" for the points; by only keeping G_x in Equation (2), we recover a purely random sampling strategy. Biased sampling with w_x as the sampling weight is exactly equivalent to maximizing the perturbed log-probabilities; thus combining and balancing the informativeness signal with randomness. Thus, HAL utilizes the signals captured in w_x to guide the selection towards more valuable points, but also leverages randomness to resist the noise and variance in the informativeness signal.

5.2 Cost-weighting Decisions

There are three challenges in deriving the sampling weight (the function $\mathcal{P}$) for HAL: ❶ Decide which informativeness measure to use. ❷ Combine the informativeness and the cost. ❸ Adapt the sampling weight to different budgeting interfaces and cost types that the ADCP needs to support.

Informativeness: There are many ways in AL literature to capture the informativeness, as discussed in Section 2.4. Nevertheless, most of these methods incur additional constraints on the ML task, ML model, or the input/output, which limits the strategy's applicability in the various scenarios that the ADCP needs to support. For example, an informativeness measure like *expected model change* requires the model to be differentiable to compute the gradient [57, 60], which many ML models (e.g., random forests) do not satisfy.

Given this, we use a simple but more accessible informativeness signal for HAL: *uncertainty* $u(x)$ [36]. A higher $u(x)$ corresponds to a higher probability of incorrect prediction for x. $u(x)$ is directly available among many common ML models, such as random forests and logistic regression. There are also separate methods to derive $u(x)$ for a wide range of ML models/tasks, such as ensemble, quantile regression, and confidence calibration [22, 46, 47, 69]. In a classification task, $u(x)$ can be the probability that x does not belong to the predicted class. In a regression task, $u(x)$ can be the output variance of x's prediction. Uncertainty-based AL exploits the correlation between points with high uncertainty and the probability of a prediction error. Thus, acquiring labels with higher uncertainty is likely to give more information and improve the ML model's prediction after retraining.

Combining Cost: To combine the uncertainty $u(x)$ and the cost $c(x)$, previous works have directly summed the impact of $u(x)$ with $c(x)$ (or $-c(x)$) with potential reweighting [30, 32]. The issue, however, is that $u(x)$ and $c(x)$ fundamentally come from two different domains. The two metrics can have drastically different scales as we generalize across different databases and it is intricate to derive a proper reweighting.

We use a different approach to combine $u(x)$ and $c(x)$ by dividing $u(x)$ by $c(x)$:

$$w_x = \frac{u(x)}{c(x)} \tag{3}$$

which is referred in the AL literature as Return On Investment (ROI) [25, 67]. Intuitively, ROI characterizes the amount of information in a data point per "cost unit". ROI is in proportion to $u(x)$ and in reverse proportion to $c(x)$, regardless of what the scales that $u(x)$ and $c(x)$ have. And as analyzed in Section 5.1, using the per-cost-unit uncertainty as the sampling weight for a biased sampling procedure is equivalent to a greedy optimization algorithm that maximizes the perturbed log uncertainty of the sampled points. We also investigate other variants of ROI, such as smoothing the distribution of $u(x)$ or $c(x)$ before calculating their ratio. However, we find that Equation (3) is the most natural approach to calculate the per-cost-unit uncertainty and yields the best performance.

Budgeting Interfaces and Cost Types: We provide two options for budgeting: the total estimated cost of labeling or the number of plans to execute.

For the set of database applications and ML tasks we focus on in ADCP (Section 2.1), the labeling cost comes from two sources: creating the configuration to execute a query and executing the query. The budget can be based on the estimated cost if such information for creating physical configurations and executing queries is available. Because creating a physical configuration or an index can benefit labeling multiple queries in the target dataset, when computing the cost of labeling in the sampling weight, we amortize the cost of creating an index over all the unlabeled plans that refer to this index. When the ADCP chooses to implement the configuration, it deducts the unamortized cost of creating indexes from the budget. We can also use the number of plans as the budget if the user prefers or if the estimated cost information is not available. In such a case, we remove $c(x)$ from the cost in Equation (3) regardless of the cost type (i.e., $w_x = u(x)$).

5.3 Redundancy Rejection Decisions

We experimented with state-of-the art approaches to reject selecting redundant points. Experimentally, none of these approaches showed substantially different behavior. We outline the options we explored; discovering more effective ways to reduce redundancy is an interesting avenue for future work. Capturing the redundancy in a batch of points (function $\mathcal{R}$) is challenging because there are multiple weak indicators:

Similar Selected Points: The most direct indicator for redundancy is whether there have been similar points already selected. This is a weak indicator because similar points may have disparate labels in databases. For example, two plans with similar structures and operators (thus similar features) but *small* differences in join orders or selectivities can sometimes have significantly different execution costs.

Density: Since a data point's label does not necessarily generalize to all its neighbors, we observe that the model's target data prediction can benefit from labeling more points in high-density regions. Labels in dense regions might generalize to more number of related data points.

Uncertainty: Since $u(x)$ corresponds to the probability of incorrect prediction, we observe that regions with higher $u(x)$ tend to require more labels for the model to generalize to all the points. This is also a weak indicator because $u(x)$ can be imprecise before model retraining.

HAL combines these indicators leveraging a clustering-based approach. Previous AL strategy has used clustering to capture the distribution of the target data explicitly, and select the same number of points from each cluster in a round-robin fashion [62]. The intuition is that points within a cluster are similar, and thus redundant. A direct application of this approach does not capture the density and uncertainty indicators. Given that, we propose two novel adaptations:

Sub-modular Threshold: Selecting the same number of points from each cluster does not give preference to larger clusters (regions with higher density). Within each cluster, there is also a diminishing return as more points are selected. Thus, HAL uses a monotone concave function $\gamma(n) = n^a$, where $a \in (0, 1)$ and n is the cluster size, to restrict the maximum number of points to select from each cluster. $\gamma(n)$ is larger for larger clusters, and the concavity naturally captures the diminishing return [34]. HAL sets a initially to be small. HAL incrementally multiplies a if all clusters reach the threshold: $\gamma(n) = n^a, n^{2a}, n^{3a}...$ until $\gamma(n) >= n$. Thus, the threshold automatically adapts to larger budget sizes.

Uncertainty Clip: Since regions with higher $u(x)$ tend to require more labels, HAL clips redundancy rejection for high-uncertainty clusters. More specifically, HAL does not enforce the threshold γ if a cluster's average $u(x)$ is higher than a percentile β among all the target data uncertainties.

There are two clustering details. (1) Similarity: We first perform quantile transformation on the data [19] and then use the L_2 metric. We observe this approach performs better than directly applying standard metrics, such as L_2 and cosine, because it brings different feature dimensions into the same scale. (2) Algorithm: We investigate a few common clustering algorithms and observe similar performance with properly tuned parameters [16, 41, 43]. We use DBSCAN for ADCP as it has moderate cost and is less sensitive to hyper-parameters across database workloads (Section 7.2).

In general, our redundancy rejection mechanism is conservative with the sub-modular threshold and uncertainty clip adaptations. Though missing certain rejection opportunities, our design meets the ADCP's robustness requirement to support various applications (see analysis in Section 7.8).

Another alternative to combine the weak indicators is to integrate them into the "value score" of the points. Then an AL strategy can either maximize the value score [73] or use the score as the sampling weight w_x [9]. However, the complex interactions between these weak indicators can make the value score brittle and more unreliable, which leads to poor performance in our evaluation (Section 7.4).

6 HAL EXECUTION EXAMPLE

We now illustrate the execution of HAL using a running-example, as shown in Figure 3. Given the target test data (5 points in total), the base ML model, and the labeling cost budget (6) specified by the user, HAL executes the following steps. ❶ Prepare the necessary information by deriving the sampling weight w_x using $u(x)$ and $c(x)$ (Equation (3)) and perform clustering on the target data (Section 5.3). ❷ Initialize the algorithm's relevant states, including the selected points (None), the qualified points (all 5 points), and the redundant points (None) as defined in Algorithm 1. HAL maintains the set of redundant points by tracking the IDs of the clusters that reach the rejection threshold. ❸ Sample one point with a cost 4 according to weight w_x. HAL updates the remaining budget to 2. It calculates the qualified points (2 points) under the new budget, and the clusters (1 cluster) that reach the rejection threshold. ❹ Sample a second point with a cost 2. After this step, there are no qualified points since the budget is 0. There are also two redundant clusters given the selected points. ❺ Return the selected points and terminate, since the budget has been exhausted.

7 EXPERIMENT

We couch our evaluation on the ADCP's application and usage scenarios discussed in Section 3.2. We use two ML tasks (Section 2.1) to evaluate the ADCP under different budget sizes, cost types, and budgeting interfaces (Sections 7.4 to 7.7) We also use a variety of industry-standard benchmarks and customer workloads. Our evaluation shows that the ADCP significantly reduces the ML model's prediction error on the target data in all scenarios (e.g., over 75% error reduction with approximately 100 query executions).

In each scenario, we compare the ADCP with HAL against a number of baseline AL strategies, ranging from straw man approaches to advanced techniques in the AL literature that

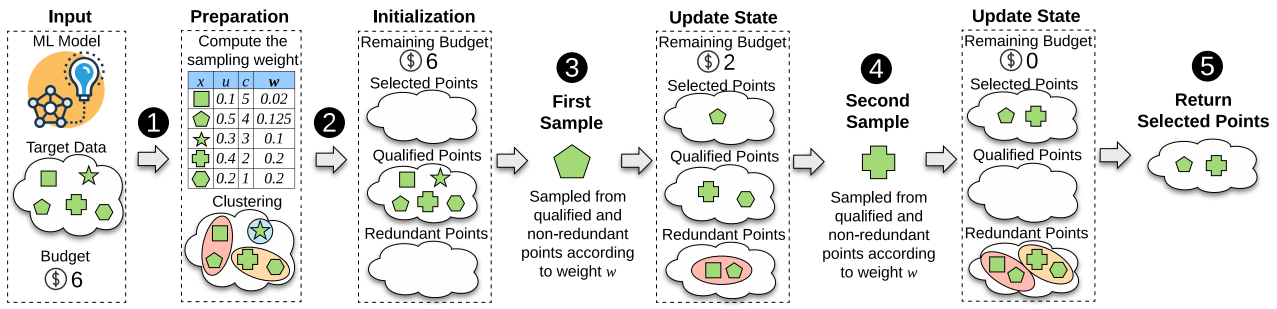


Figure 3: HAL Execution Example – HAL selects two points to label under the budget (details explained in Section 6).

are more aligned with the ADCP's requirements. Unlike HAL, all the baselines only partially address the ADCP's holistic AL challenges (Section 3.3), which we annotate in detail in Section 7.3. *The results show that HAL significantly outperforms all the baselines in all scenarios*, and no baseline performs well in all cases. This demonstrates HAL's superiority as a robust, cost-sensitive, and batch-friendly strategy. Under the same budget, HAL reduces the prediction error by up to 2× more compared to all the baselines.

We also analyze HAL and the baselines' properties in-depth in Section 7.8. This analysis further reflects how HAL better handles ADCP's challenges compared to the baselines.

7.1 Evaluation Protocol

Data Generation: We use the same methodology discussed in Section 2.2 to generate the target data for each held-out database, which rotates among all the 14 database workloads (Section 7.2). We construct the target data for both ECP and PRP tasks by executing the queries from our workloads. We use a state-of-the-art index tuner [13] to recommend a set of indexes for each query, and enumerate subsets of the recommended indexes as the index configurations. The optimizer then generates candidate plans under each index configuration for a query. We implement the different index configurations, execute the queries in isolation, and record the plans' execution costs (e.g., CPU time) for labeling. For ECP, the collected plans and their execution costs are the target data and the labels. For PRP, we construct plan pairs from all the plans of the same query as the target data, and assign labels based on whether the first plan is more expensive than the second. Although we use the same execution data for both ECP and PRP, the two tasks use distinct featurizations for their inputs. Thus, the two tasks' data distributions after featurization are substantially different. When the ADCP labels a pair for PRP, it executes both plans. Table 2 summarizes the statistics of the collected plans and indexes.

Experiment Execution: We run an active data collection experiment for each target data with three repetitions, and report the average result from all the experiments. By default, we specify the data collection budget by the labeling cost, which includes both the index creation and the plan execution costs. In each experiment, we first train a base ML model ("cold-start" model) using the labeled data from all databases except the held-out one. We set the total budget spent by the ADCP as 150× the average estimated plan execution cost (avg($c(x)$)) in the target data, which is on average 18% of the entire target data's labeling cost. We use the optimizer's cost model as the crude cost estimator, which does not require any additional training data. To evaluate how the ADCP performs with different model retraining frequencies and budget sizes, we evenly split the total budget among multiple (3-15) iterations. In each iteration, we (1) use ADCP to label a batch of data points under the divided budget, (2) retrain the ML model with the existing and the newly labeled data, and (3) evaluate the model's prediction on the target data.

Metrics: For ECP, we use L_1 error on the target data to measure the model's performance as used in the previous work for this task [37]. We normalize the L_1 error between $[0, 1]$ for clarity. For PRP, we use F1 error on the target data as the performance metric [7], which is robust to skew in the distribution of classes where prediction accuracy is susceptible. The F1 error is between $[0, 1]$. We use the error metrics on the entire target data since this reflects how databases use ML models in reality. When we evaluate the model's prediction on the remaining target data, excluding the data points labeled by the AL strategy, we observe similar results.

7.2 Implementation Details

Workloads: We use a diverse collection of both industry-standard benchmarks and real customer workloads under a variety of data sizes, distributions, and query complexity. Table 2 summarizes the key statistics of the workloads. For the TPC workloads, we use two different scale factors, 10 and 100, which share query templates but use different parameters and have different data sizes and distributions. We also use a skewed data generator [52] for TPC-H to make the execution cost estimation more challenging. All queries in these workloads are SELECT statements.

ML Tasks: We use the following featurizations and models.

Workload	DB size (GB)	# tables	# queries	Avg. # joins	Total # Plans	# Plan Pairs	# in-dexes
TPC-DS 10g	11.2	24	92	7.9	3, 500	200, 825	329
TPC-DS 100g	87.7	24	92	7.9	3, 714	211, 541	359
TPC-H 10g Zipf	12.1	8	22	2.8	299	6, 986	60
TPC-H 100g Zipf	132	8	22	2.8	306	6, 600	57
Customer1	87.7	20	111	5.9	4, 669	144, 474	389
Customer2	1723	23	34	7.2	2, 364	214, 842	185
Customer3	44.6	614	32	8.1	584	17, 926	83
Customer4	1.2	8	125	1.6	2, 539	153, 752	134
Customer5	283	3, 394	35	7.2	2, 041	133, 671	147
Customer6	9.9	474	311	21	3, 727	173, 025	248
Customer7	93	22	23	5.2	841	37, 157	128
Customer8	0.25	129	474	1.1	3, 746	82, 738	287
Customer9	17.0	32	10	8	242	7, 854	37
Customer10	7.2	81	399	0.8	2, 820	19, 257	228

Table 2: Workloads – Aggregate statistics about the schema and query complexity for the read-only workloads.

Plan regression prediction (**PRP**): We use the same method in a recent work for this task to featurize a plan [14], which includes the operator types (e.g., Index Scan), the execution modes (e.g., single-threaded/parallel), each operator's estimated work, and the plan structure. We also use the same plan-pair featurization technique, which calculates the attribute-wise difference between the two plans' features. We use random forest (RF) [38] as the default ML model since it has high accuracy and low training cost [14]. We use 100 learners (trees) and at least 1 sample in any leaf node set through cross-validation. We use early stopping to prevent node splitting if the impurity (see [7]) is below 10^{-6}.

Execution cost prediction (**ECP**): To unify the evaluation protocol, we use the same technique as PRP to featurize a plan and also use RF as the ML model. We use 200 learners (trees) and at least 5 samples in any leaf node set through cross-validation. We use early stopping to prevent node splitting if the impurity is below 10^{-6}.

We also evaluate the ADCP under other featurizations and ML models (e.g., LightGBM [31]) for PRP and observe similar results; we omit these due to space limitations.

Software and Hardware: We implement the ML models using `scikit-learn` and `LightGBM` in `Python`. We train the ML models on a virtual machine with 144 GB RAM and 72 virtual CPU threads based on the Intel Xeon Platinum 8168 (SkyLake) processor. In our experiments, training the ML model on average takes five minutes for both ML tasks.

Hyper-Parameters: We use cross-validation to set the hyper-parameters for the redundancy rejection mechanism (Section 5.3). For ECP, we set the DBSCAN's neighbor distance as 0.1 and the minimum samples in a core point's neighborhood as 5 for all workloads. For PRP, we set them as 1 and 4, respectively. We set the uncertainty clip percentile β as 60% and 70% for ECP and PRP, respectively. And we set the submodular threshold parameter a as 0.1 for both tasks.

Method	Uncertainty	Cost Estimate	Redundancy	Randomness
HAL	✓	✓	✓	✓
Rand				✓
Cheap		✓		
Index				✓
Uncertain	✓			
Hybrid	✓			✓
RBMAL	✓		✓	
Round	✓	✢	✓	
Similar		✢	✓	
OPT	N/A	N/A	N/A	N/A

Table 3: AL Baseline Categorization – Summary on how the AL strategies leverage different signals and sampling options to address the holistic challenges in Section 3.3[1].

7.3 Baselines

We compare our HAL strategy with nine baselines based on either heuristics or AL literature. Table 3 categorizes how the baselines use different AL signals and selection approaches. Our baselines include:

Rand: Randomly select data points from the target data. As discussed in Section 5.1, Rand is a simple but surprisingly effective strategy that can outperform advanced AL strategies in many cases [48]. Rand is robust to unreliable AL signals, e.g., the informativeness and the cost estimates.

Cheap: Select data points with the lowest cost $\hat{c}(x)$. Cheap is cost-sensitive, and maximizes the number of labeled points.

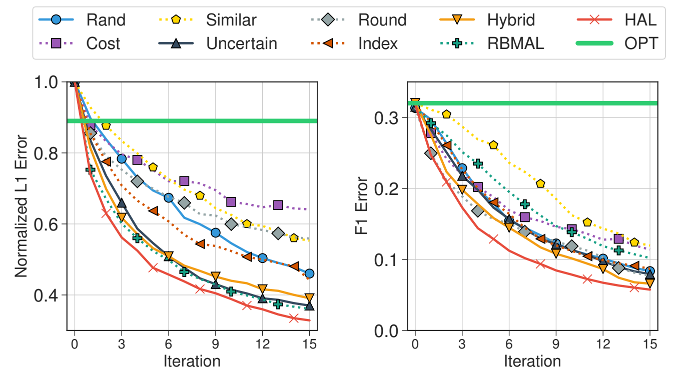
Index: Sample queries with a fixed weight p for queries that use indexes, and with the weight $1 - p$ for the other queries. Index is implicitly cost-sensitive without a cost estimator. We tune p to 0.4 where Index performs the best.

Uncertain: Select data points with the highest uncertainty $u(x)$ [36] to maximize the newly labeled points' total uncertainty. We also evaluate a variant of this strategy that selects points ordered by $\frac{u(x)}{c(x)}$. Because $c(x)$ has a wider and more skewed distribution than $u(x)$ in many workloads, this variant ends up similar to the Cheap baseline. Since the variant performs worse than Uncertain on average, we do not include it in our baselines.

Hybrid: This strategy uses Uncertain to select points with a fixed portion of the budget (tuned to 0.5), and it uses Rand for the remaining budget. Hybrid is a batch-friendly AL strategy used in database crowdsourcing [24].

RBMAL: RBMAL is a state-of-the-art batch-friendly AL strategy [9] that augments Uncertain with a weighted distance of each data point to the labeled training data. The weight for the distance degrades as more points are selected. Empirically, we found that calculating the weight and the distance to the entire training set performs better than only to the labeled/selected points in the target data. Specifically, RBMAL selects points in order sorted by $w_x = u(x) + \alpha *$

[1]We use ✓ to denote that the original strategy has that property, and ✢ to denote that we extended the strategy to have that property in our evaluation.



(a) ECP: 10× Avg Cost/Iteration **(b)** PRP: 10× Avg Cost/Iteration

Figure 4: Baseline Performance – The performance improvement on the ML models with baseline strategies.

$dis(x, X_L \cup X_A)$, where $dis(x, X_L \cup X_A)$ is a distance metric between x and the labeled training data.

Round: Cluster the target data, then select one point from each cluster in a round-robin fashion ranked by the points' "informativeness score". This is a state-of-the-art AL strategy that considers redundancy and is batch-friendly [62]. Round can be cost-sensitive by using the HAL's sampling weight as the score, and yields better performance.

Similar: Define w_x for each point as the difference between the point's distances to the labeled training data and the unlabeled target data, and select points in sorted w_x order. Similar is batch-friendly and captures the redundancy [73]. We further divide w_x by $\log c(x)$: $w(x) = \frac{dis(x, X_L \cup X_A) - dis(x, X_U)}{\log c(x)}$. This makes Similar cost-sensitive and yields better results.

OPT: A crude baseline that directly derive the labels for the ML tasks through the optimizer's cost estimation. We assign labels for PRP by comparing the optimizer's costs for the plan pair. To derive ECP's labels, we use Huber regression [28], which is robust to outliers, on the labeled training data with the optimizer's cost as input. OPT does not require any additional label or model retraining.

7.4 Baseline Improvements

We first evaluate ADCP with all the baseline strategies. For both ML tasks, we run 15 continuous iterations of active data collection with a 10× avg($c(x)$) budget for each iteration. Figure 4a and Figure 4b show ECP's normalized L_1 error and PRP's F1 error (discussed in Section 7.1) on the target data after each iteration.

Directly using the optimizer's estimate (OPT) performs similar or better than the ML models without additional labels in the target data (iteration 0). This reinforces the need for active data collection to improve the model's predictions.

Maximizing the number of labeled points (Cheap), which is arguably the most simple and intuitive strategy, fails to deliver reasonable model error reduction; this is because it may select many uninformative and redundant points.

The two more sophisticated AL strategies in literature (Similar and Round) do not achieve satisfactory error reduction, either. The informativeness signal can be unreliable. Combining other signals related to cost and redundancy aggravates the noise and variance in the informativeness score, and make these deterministic selection methods brittle.

Rand, which is robust to unreliable signals, outperforms Similar and Round in most cases. However, the error reduction is limited since Rand is oblivious to the informativeness values. Index adds a naïve cost signal to Rand but does not bring much performance improvement either.

The *uncertainty-based* strategies generally have reasonable performance. Although Uncertain only uses the informativeness signal, it outperforms many advanced strategies. Hybrid, which combines Uncertain and Rand, outperforms both of them. This confirms our intuition that randomness can robustify AL under noisy signals. RBMAL adds a more complex distance metric to Uncertain, but performs similar (ECP) or worse (PRP) than Uncertain. This further demonstrates the problem's difficulty and a complex strategy that partially addresses the challenges in Section 3.3 may perform worse than a simple approach.

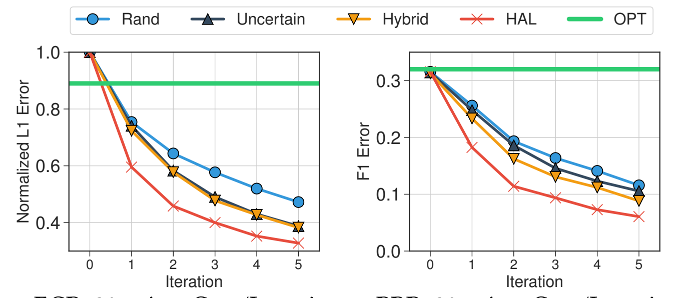
HAL, which holistically addresses ADCP's AL challenges, significantly outperforms the baselines. It especially reduces the model prediction error with small budgets. For example, with 10× avg($c(x)$) budget (iteration 1), it reduces ECP's L_1 error by 27% and PRP's F1 error by 24%.

We further evaluate the baseline strategies in a wide range of settings, such as different budget sizes or cost types. Uncertain and Hybrid are the best performing baselines across all settings. Hence, for the remaining results, we use these two strategies along with the rudimentary baseline, Rand, as representatives for comparing with HAL.

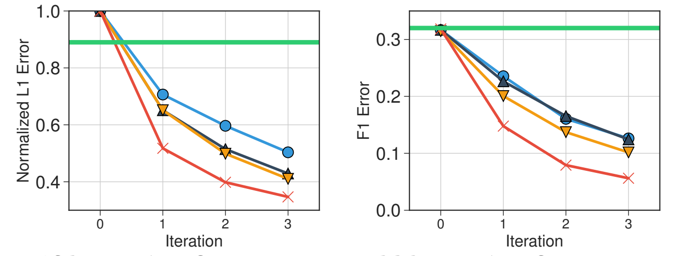
7.5 Budget Sizes

We now evaluate the AL strategies for the ADCP with different budget sizes per iteration, and subsequently different number of iterations. Results in Figure 5 show that HAL significantly outperforms all the other baselines.

Many ML models can be expensive to retrain. However, the retraining cost might not be directly comparable with the plan execution costs. Thus, we use the experiments in this section and Section 7.4 to evaluate how the ADCP performs with cheap/expensive models with different retraining frequencies. The results show that HAL performs well even when the retraining frequency (number of iterations) is low. For example, with an 100× avg($c(x)$) budget, HAL reduces ECP's L_1 error by 61% with both 10 iterations (Figure 4a) and 2 iterations (Figure 5a). In contrast, Hybrid and Uncertain,



(a) ECP: 30× Avg Cost/Iteration **(b)** PRP: 30× Avg Cost/Iteration



(c) ECP: 50× Avg Cost/Iteration **(d)** PRP: 50× Avg Cost/Iteration

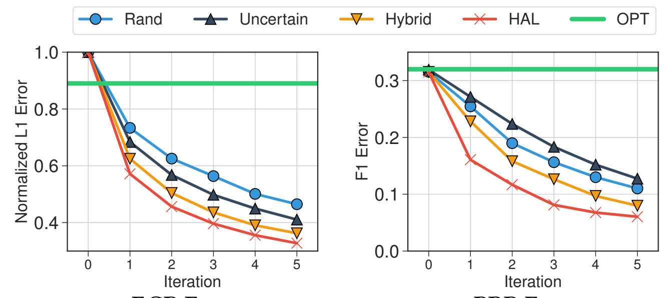
Figure 5: Budget Sizes – The improvement on the ML models with different ML tasks and budget sizes.

which are the best performing baselines, reduces ECP's L_1 error by ≈60% with 10 iterations (Figure 4a) and only 50% with 2 iterations (Figure 5a). Since Hybrid and Uncertain do not reason about redundancy, they select more redundant points with larger batch sizes. We observe similar trends for PRP in Figure 4b and Figure 5b.

Since users are often cost-sensitive, they may decide to terminate the data collection sooner with smaller resource cost and model retraining time. Thus, improving the model's prediction on the target data in the early iterations is important. The results in Figure 5 show that after the first iteration, HAL reduces PRP's F1 error or ECP's L_1 error 1.5×-2× compared to all the baselines under two different budget sizes. Note that the labeling resource cost varies with the budget size when the iteration count is the same. These results show that HAL efficiently improves the model's target data prediction with small resource budget and model retraining iterations.

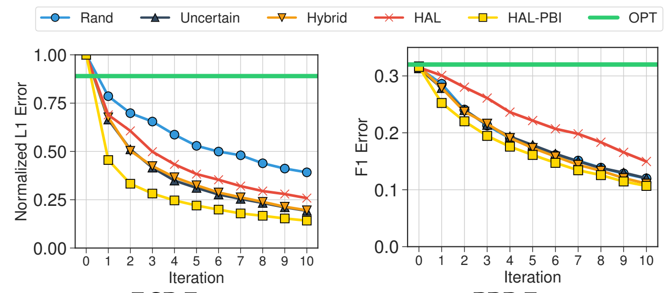
7.6 Cost Type

The ADCP supports database applications with different cost types. We have so far focused on the index tuner where the labeling cost of points includes both the plan execution and the index creation cost. We now consider a different application, query optimization, where the optimizer estimates the plan execution costs in the current index configuration. In such an application, the labeling cost does not contain the index creation cost. Thus, we evaluate the ADCP's performance with the index cost excluded from the cost estimation $c(x)$. We set the budget for each iteration as 30× avg($c(x)$).



(a) ECP Error **(b)** PRP Error

Figure 6: Cost Type – The model's performance improvement when the active data collection only includes the plan execution cost, but not the index creation cost.



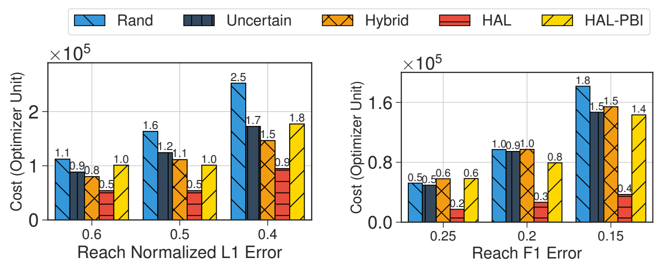
(a) ECP Error **(b)** PRP Error

Figure 7: Model error reduction under Plan Budgeting Interface – Each iteration has a budget on the number of plans that the AL strategy can select regardless of the cost.

The results in Figure 6 show that HAL significantly outperforms the baselines with the optimizer's cost type. Especially, HAL reduces PRP's F1 error at least 2× compared to all the baselines after the first iteration (Figure 6b). The change in the cost affects how the strategy uses the budget, and the results show that HAL handles both cost types well. Note that Hybrid performs worse for the index tuner cost type (e.g., Figure 5a vs. Figure 6a) because cost-oblivious strategies struggle to reason about the interdependent index cost.

7.7 Budgeting Interface

We now evaluate the ADCP by a different budgeting interface, i.e., by the number of executed plans, as discussed in Section 3.2. We set the ADCP to execute 2% of the target data's total number of plans in each iteration, with the remaining setup the same as Section 7.1. We denote our adapted sampling strategy for this plan budgeting interface as **HAL-PBI**, where we remove the denominator $c(x)$ from the computation of the sampling weight w_x (Section 5.2). HAL-PBI is especially useful when a good cost estimator is unavailable. Besides evaluating how well HAL-PBI reduces the model's prediction error, we also study (1) how much resource cost the cost-oblivious HAL-PBI spends, and (2) assuming a cost estimator is still available, how **HAL** performs.



(a) ECP Cost - Error Target **(b)** PRP Cost - Error Target

Figure 8: Costs under Plan Budgeting Interface – The cost (in the optimizer's unit) to reach certain error targets.

The results in Figure 7 show that HAL-PBI outperforms all the baselines after the same number of iterations. Especially, it reduces the prediction error 1.6×-2.2× after the first iteration compared to the baselines for both tasks. Since the strategies can spend different amount of costs in an iteration, we also measure the total cost spent by each strategy to reduce the model's prediction error to certain targets. Figure 8 shows that HAL-PBI spends similar or lower cost for the data collection to achieve the same error target compared to the baselines. This demonstrates that HAL-PBI is robust and batch-friendly even without accurate cost estimation.

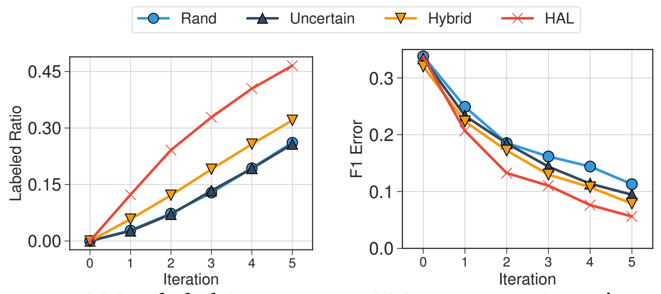
The results in Figure 8 also show that if a reasonable cost estimator is available, the cost-sensitive HAL spends significantly less cost than other strategies to reach a fixed error target. For example, HAL uses at most 0.35× cost to reduce PRP's F1 error to 0.15 compared to all the other strategies (Figure 8b). However, HAL may also select points that are only moderately informative when the cost is low. Thus, with the same number of labeled points, HAL can yield less total information. The results in Figure 7 show that HAL requires many more iterations to achieve the same error target compared to the baselines, which is undesirable when the ML model has high training cost (e.g., for DNNs or SVMs).

HAL and HAL-PBI show different advantages over the baselines in this scenario, depending on the ratio between the ML model retraining cost and the labeling cost. Balancing this tradeoff is an interesting avenue for future work.

7.8 Additional Analysis

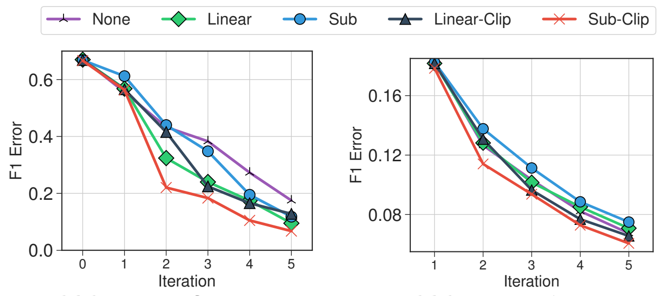
We now conduct a deeper analysis on HAL's performance compared to other strategies. We focus on the PRP task with the default setup (Section 7.1), and set the budget per iteration to 30× avg($c(x)$); we observe similar results in other setups.

We first measure the ratio between the number of labeled points by the ADCP and the target data's total number of points. Figure 9 shows that HAL labels more points than the baselines at each iteration, which suggests that the average cost of HAL's labeled points is lower. This demonstrates that HAL is cost-sensitive and selects cheaper points.



(a) PRP Labeled Ratio **(b)** PRP Expensive Points' Error

Figure 9: Strategy Properties – The ratio of the labeled points and the prediction error on the expensive points.



(a) PRP Error - Customer8 **(b)** PRP Error - Average

Figure 10: Redundancy Rejection Design Space – Performance with different options in the design space of the redundancy rejection mechanism.

Since HAL on average selects cheaper points than the baselines, we then investigate whether HAL reduces less error of the model's predictions on the more expensive points. This is important because expensive points may have higher performance impact to the DBMS than cheaper ones, such as consuming more resource or resulting in longer query latency. We measure the ML model's prediction error on the points with a cost $c(x)$ higher than the median cost in the target data. Figure 9b shows that HAL still yields lower prediction error on the expensive points compared to the baselines, which indicates that HAL does not sacrifice the label's value (informativeness) while selecting points.

Finally, we examine the design space of the redundancy rejection mechanism (Section 5.3). We denote the strategy that excludes the redundancy rejection from HAL as *None*. On top of None, we denote the direct adaptation of the clustering-based approach that samples one point from each cluster in turn as *Linear*. We denote the submodular threshold adaptation for Linear as *Sub*. We add the uncertainty clip adaptation on top of both Linear and Sub, denoted as *Linear-Clip* and *Sub-Clip*. Sub-Clip is equivalent to HAL.

We observe that Sub-Clip performs significantly better compared to None on a few workloads, with an example shown in Figure 10a. This confirms the importance of capturing the redundancy in the AL strategy. In many workloads, however, Sub-Clip has similar performance with None. Figure 10b shows the average error reduction for different rejection mechanisms, where we zoom in the y-axis and omit iteration 0 for clarity. Sub-Clip has a moderate average advantage over None, and does not perform worse than None on any workload. Other rejection mechanisms either perform similar or worse than None. This is because (1) capturing the redundancy is challenging (Section 5.3), and (2) None already spreads the samples across different regions, which reduces the potential redundancy in a batch of selected points.

8 RELATED WORK

ML models have been used to enhance several aspects of databases, like query run-time prediction [18, 37], query optimization [35, 44, 50], self-administration [39, 51], physical design [5, 61], knob configuration [66], and index recommendation [14]. Their performance typically degrades when they serve a production workload that was unseen during training time; (see [14] for a case study in index recommendation; and [37] for an example in query run-time prediction).

One solution to combat this performance degradation is to passively collect telemetry from the production workload [63]. Although this approach does not incur extra execution overhead, the database does not control when or what plans to execute. Our solution, in contrast, employs active learning [57]. Active learning is known to be more sample-efficient than passive supervised learning in many problem settings [62, 73], which can mean that ML models are re-trained on collected data in a more cost-effective way.

Active Learning is a vast field, but many tailored AL solutions do not work in the database setting. For example, some works require the models to be differentiable [4, 8, 57, 60], which do not apply to many ML models such as random forests. Other methods require a large amount of retraining [23, 40, 72], rendering them computationally prohibitive for many complex ML models.

Most database applications, including ours, require batch data collection. Batch AL is well studied (e.g., see [9]), and aims to find datapoints that maximize model uncertainty as well as diversity [4]. We compare to the state-of-the-art techniques (e.g., by explicitly capturing the data distribution [49, 62] or applying density weighted methods [58, 73] or ranking [9]) in our experiments, and incorporate simple notions of uncertainty and diversity into HAL.

Several AL strategies are also cost-sensitive [59]. Some techniques use return on investment to guide labeling effort [25, 67], which HAL draws inspiration from. Recent benchmarks [75] also show that uniform random sampling is a formidable baseline and uncertainty sampling [55, 56, 68] is a useful heuristic, especially when the data is noisy [48].

Robustness (or, AL with distribution shift) is the key challenge in our work that is relatively understudied in the literature. Related approaches compose domain adaptation techniques with active learning [53, 64]. Other approaches use transfer learning [70, 71]. The key idea is to adapt the initial ML model to hopefully provide better signals in situations when domain adaptation/transfer learning can succeed. In our problem settings, the mismatch in training and target is so severe (e.g., see Figure 1) that these techniques (like covariate shift correction) are doomed to fail.

There is a long and successful history of using active learning in database systems for tasks like crowdsourcing [24, 47] or data mining [3, 27, 74]. These works use human labeling to improve the performance of the front-end applications. However, in these settings labels are noisy and signals/costs are accurate. In our setting of AL with distribution shift, the labels we collect are accurate and signals/costs are noisy. This requires us to incorporate mechanisms in HAL to remain robust to noisy and varying signals using Gumbel sampling.

9 CONCLUSION AND FUTURE WORK

The training-test data mismatch can significantly limit the applicability of ML techniques in databases, especially in production deployments. We propose an active data collection platform (ADCP) to address this issue by collecting more labeled data with extra resources. We formulate an AL problem to collect labels for specific target test data, and design a simple yet effective AL strategy, HAL, that is robust to unreliable signals, cost-sensitive, and batch-friendly. Empirically, ADCP greatly improves the ML model's performance with small resource budget, and HAL significantly outperforms state-of-the-art AL baselines in a wide range of scenarios. We contend that this is a crucial step towards building a complete ML pipeline to enhance database systems.

There are several interesting future directions for this work. For example, ADCP may jointly learn a labeling cost estimation model with the newly labeled points. There is also the opportunity to combine the additionally collected labels for the individual production workload into a global model with better generalization. Our early investigations also show that adjusting the training weight after collecting more labels, like covariate shift correction, can further benefit the model error reduction on the target data. We view this work as the first step towards tackling the holistic AL problem that arises from applying ML to complex database systems.

10 ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers and Andy Pavlo for their valuable feedback.

REFERENCES

[1] Deepak Agarwal, Lihong Li, and Alexander Smola. 2011. Linear-time estimators for propensity scores. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 93–100.

[2] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollar, Arun Marathe, Vivek Narasayya, and Manoj Syamala. 2005. Database tuning advisor for microsoft sql server 2005. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of data*. ACM, 930–932.

[3] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 783–794.

[4] Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. 2019. Deep batch active learning by diverse, uncertain gradient lower bounds. *arXiv preprint arXiv:1906.03671* (2019).

[5] Debabrota Basu, Qian Lin, Weidong Chen, Hoang Tam Vo, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. 2015. Cost-model oblivious database tuning with reinforcement learning. In *International Conference on Database and Expert Systems Applications*. Springer, 253–268.

[6] Steffen Bickel, Michael Brückner, and Tobias Scheffer. 2009. Discriminative learning under covariate shift. *Journal of Machine Learning Research* 10, Sep (2009), 2137–2155.

[7] Christopher Bishop. 2006. Pattern Recognition and Machine Learning. *Pattern Recognition and Machine Learning* (2006).

[8] Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. 2005. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research* 6, Sep (2005), 1579–1619.

[9] Thiago NC Cardoso, Rodrigo M Silva, Sérgio Canuto, Mirella M Moro, and Marcos A Gonçalves. 2017. Ranked batch-mode active learning. *Information Sciences* 379 (2017), 313–337.

[10] 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*. ACM, 34–43.

[11] Surajit Chaudhuri and Vivek Narasayya. 2007. Self-tuning database systems: a decade of progress. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 3–14.

[12] Benoit Dageville, Dinesh Das, Karl Dias, Khaled Yagoub, Mohamed Zait, and Mohamed Ziauddin. 2004. Automatic SQL tuning in oracle 10g. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 1098–1109.

[13] Sudipto Das, Miroslav Grbic, Igor Ilic, Isidora Jovandic, Andrija Jovanovic, Vivek R Narasayya, Miodrag Radulovic, Maja Stikic, Gaoxiang Xu, and Surajit Chaudhuri. 2019. Automatically indexing millions of databases in microsoft azure sql database. In *Proceedings of the 2019 International Conference on Management of Data*. 666–679.

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

[15] Bailu Ding, Sudipto Das, Wentao Wu, Surajit Chaudhuri, and Vivek Narasayya. 2018. Plan stitch: harnessing the best of many plans. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1123–1136.

[16] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96.

[17] 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), 44.

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

[19] Warren Gilchrist. 2000. *Statistical modelling with quantile functions*. CRC Press.

[20] Ian Goodfellow, Aaron Courville, and Yoshua Bengio. 2016. Deep learning. The MIT Press.

[21] Emil Julius Gumbel. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*. Number 33. US Govt. Print. Office.

[22] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On Calibration of Modern Neural Networks. In *International Conference on Machine Learning*. 1321–1330.

[23] Yuhong Guo and Russ Greiner. 2007. Optimistic active learning using mutual information. In *Proceedings of the 20th international joint conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., 823–829.

[24] Daniel Haas, Jiannan Wang, Eugene Wu, and Michael J Franklin. 2015. CLAMShell: Speeding up Crowds for Low-latency Data Labeling. *Proceedings of the VLDB Endowment* 9, 4 (2015).

[25] Robbie Haertel, Kevin D Seppi, Eric K Ringger, and James L Carroll. 2008. Return on investment for active learning. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*, Vol. 72.

[26] Shohei Hido, Yuta Tsuboi, Hisashi Kashima, Masashi Sugiyama, and Takafumi Kanamori. 2011. Statistical outlier detection using direct density ratio estimation. *Knowledge and information systems* 26, 2 (2011), 309–336.

[27] Enhui Huang, Liping Peng, Luciano Di Palma, Ahmed Abdelkafi, Anna Liu, and Yanlei Diao. 2018. Optimization for active learning-based interactive database exploration. *Proceedings of the VLDB Endowment* 12, 1 (2018), 71–84.

[28] Peter J Huber et al. 1973. Robust regression: asymptotics, conjectures and Monte Carlo. *The Annals of Statistics* 1, 5 (1973), 799–821.

[29] Thorsten Joachims. 1999. Transductive Inference for Text Classification using Support Vector Machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 200–209.

[30] Ashish Kapoor, Eric Horvitz, and Sumit Basu. 2007. Selective supervision: guiding supervised learning with decision-theoretic active learning. In *Proceedings of the 20th international joint conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., 877–882.

[31] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. 3146–3154.

[32] Ross D King, Kenneth E Whelan, Ffion M Jones, Philip GK Reiser, Christopher H Bryant, Stephen H Muggleton, Douglas B Kell, and Stephen G Oliver. 2004. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature* 427, 6971 (2004), 247.

[33] Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Stochastic Beams and Where To Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement. In *International Conference on Machine Learning*. 3499–3508.

[34] Andreas Krause and Carlos Guestrin. 2008. Beyond convexity: Submodularity in machine learning. *ICML Tutorials* (2008).

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

[36] David D Lewis and Jason Catlett. 1994. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning Proceedings*

1994. Elsevier, 148–156.

[37] Jiexing Li, Arnd Christian König, Vivek Narasayya, and Surajit Chaudhuri. 2012. Robust estimation of resource consumption for sql queries using statistical techniques. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1555–1566.

[38] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[39] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J Gordon. 2018. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 631–645.

[40] Oisin Mac Aodha, Neill DF Campbell, Jan Kautz, and Gabriel J Brostow. 2014. Hierarchical subquery evaluation for active learning on a graph. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 564–571.

[41] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[42] Chris J Maddison, Daniel Tarlow, and Tom Minka. 2014. A* sampling. In *Advances in Neural Information Processing Systems*. 3086–3094.

[43] Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[44] Ryan Marcus and Olga Papaemmanouil. 2018. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–4.

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

[46] Nicolai Meinshausen. 2006. Quantile regression forests. *Journal of Machine Learning Research* 7, Jun (2006), 983–999.

[47] Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. 2014. Scaling up crowd-sourcing to very large datasets: a case for active learning. *Proceedings of the VLDB Endowment* 8, 2 (2014), 125–136.

[48] Stephen Mussmann and Percy Liang. 2018. On the Relationship between Data Efficiency and Error for Uncertainty Sampling. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 80. PMLR, 3674–3682.

[49] Hieu T Nguyen and Arnold Smeulders. 2004. Active learning using pre-clustering. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 79.

[50] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S Sathiya Keerthi. 2018. Learning State Representations for Query Optimization with Deep Reinforcement Learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*. ACM, 4.

[51] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. 2017. Self-Driving Database Management Systems.. In *CIDR*.

[52] Program for TPC-H Data Generation with Skew. [n.d.]. Program for TPC-H Data Generation with Skew. https://www.microsoft.com/en-us/download/details.aspx?id=52430..

[53] Piyush Rai, Avishek Saha, Hal Daumé, and Suresh Venkatasubramanian. 2010. Domain Adaptation Meets Active Learning. In *Proceedings of the NAACL HLT 2010 Workshop on Active Learning for Natural Language Processing (ALNLP '10)*. Association for Computational Linguistics, USA, 27–32.

[54] Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[55] Maytal Saar-Tsechansky and Foster Provost. 2004. Active sampling for class probability estimation and ranking. *Machine learning* 54, 2 (2004), 153–178.

[56] Burr Settles. 2009. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison.

[57] Burr Settles. 2012. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6, 1 (2012), 1–114.

[58] Burr Settles and Mark Craven. 2008. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 1070–1079.

[59] Burr Settles, Mark Craven, and Lewis Friedland. 2008. Active learning with real annotation costs. In *Proceedings of the NIPS workshop on cost-sensitive learnings*. 1–10.

[60] Burr Settles, Mark Craven, and Soumya Ray. 2008. Multiple-instance active learning. In *Advances in neural information processing systems*. 1289–1296.

[61] Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. 2018. The Case for Automatic Database Administration using Deep Reinforcement Learning. *arXiv preprint arXiv:1801.05643* (2018).

[62] Xuehua Shen and ChengXiang Zhai. 2005. Active feedback in ad hoc information retrieval. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 59–66.

[63] Michael Stillger, Guy M Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO-DB2's LEarning Optimizer. In *Proceedings of the 27th International Conference on Very Large Data Bases*. 19–28.

[64] Jong-Chyi Su, Yi-Hsuan Tsai, Kihyuk Sohn, Buyu Liu, Subhransu Maji, and Manmohan Chandraker. 2019. Active Adversarial Domain Adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 1–4.

[65] Masashi Sugiyama, Neil D Lawrence, Anton Schwaighofer, et al. 2017. Dataset shift in machine learning.

[66] 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 (SIGMOD '17)*. 1009–1024.

[67] Sudheendra Vijayanarasimhan and Kristen Grauman. 2009. What's it going to cost you?: Predicting effort vs. informativeness for multi-label image annotations. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2262–2269.

[68] Andreas Vlachos. 2008. A stopping criterion for active learning. *Computer Speech & Language* 22, 3 (2008), 295–312.

[69] Stefan Wager, Trevor Hastie, and Bradley Efron. 2014. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *The Journal of Machine Learning Research* 15, 1 (2014), 1625–1651.

[70] Xuezhi Wang. 2016. *Active Transfer Learning*. Ph.D. Dissertation. BAE Systems.

[71] Xuezhi Wang, Tzu-Kuo Huang, and Jeff Schneider. 2014. Active transfer learning under model shift. In *International Conference on Machine Learning*. 1305–1313.

[72] X Xhu, J Lafferty, and Z Ghahramani. 2003. Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the ICML-2003 Workshop on The Continuum from Labeled to Unlabeled Data*. ICLM, 58–65.

[73] Zuobing Xu, Ram Akella, and Yi Zhang. 2007. Incorporating diversity and density in active learning for relevance feedback. In *European Conference on Information Retrieval*. Springer, 246–257.

[74] Zhepeng Yan, Nan Zheng, Zachary G Ives, Partha Pratim Talukdar, and Cong Yu. 2015. Active learning in keyword search-based data

integration. *The VLDB Journal-The International Journal on Very Large Data Bases* 24, 5 (2015), 611–631.

[75] Yazhou Yang and Marco Loog. 2018. A benchmark and comparison of active learning for logistic regression. *Pattern Recognition* 83 (2018), 401–415.

[76] Kai Yu, Jinbo Bi, and Volker Tresp. 2006. Active learning via transductive experimental design. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 1081–1088.

[77] 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*. ACM, 415–432.

[78] Daniel C Zilio, Jun Rao, Sam Lightstone, Guy Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. 2004. DB2 design advisor: integrated automatic physical database design. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 1087–1097.