# Maliva: **Using Machine Learning to Rewrite Visualization Queries Under Time Constraints**

Qiushi Bai, Sadeem Alsudais, Chen Li, Shuang Zhao
Department of Computer Science, UC Irvine, CA 92697, USA
{qbai1,salsudai}@uci.edu,{chenli,shz}@ics.uci.edu

## ABSTRACT

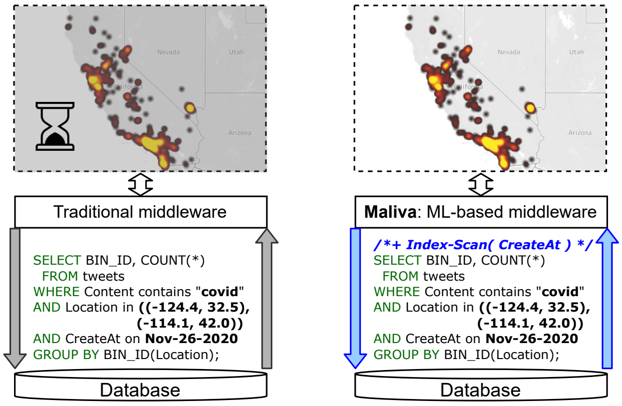
We consider data-visualization systems where a middleware layer translates a frontend request to a SQL query to a backend database to compute visual results. We study the problem of answering a visualization request within a limited time due to the responsiveness requirement. We explore optimization options of rewriting an original query by adding hints and/or doing approximations so that the total time is within the time constraint. We develop a novel middleware solution called Maliva based on machine learning (ML) techniques. It applies the Markov Decision Process (MDP) model to decide how to rewrite queries and uses instances to train an agent to make a sequence of decisions judiciously for an online request. We give a full specification of the technique, including how to construct an MDP model, how to train an agent, and how to use approximation rewriting options. Our experiments on both real and synthetic datasets show that Maliva performs significantly better than a baseline solution that does not do any rewriting, in terms of both the probability of serving requests interactively and query execution time.

## 1 INTRODUCTION

As a powerful way for people to gain insights from data quickly and intuitively, visualization is becoming increasingly important in the Big Data era. A common architecture to support data visualization has three tiers: a backend database, a middleware layer, and a user-facing frontend. The middleware translates a visualization request to a query (typically in SQL) to the database and sends the query answers to the frontend to display. This architecture is widely used due to its benefits of supporting in-situ analytics at the data source, and utilizing the database's built-in capabilities of efficient storage, indexing, query processing, and optimization. *Responsiveness* is critical in these systems [6, 10, 29], and a request needs to be served within a time budget, e.g., 500*ms*. This requirement is especially challenging when the data volume is large, and the user request has ad-hoc conditions on attributes of various types.

In this paper, we study the problem of *answering visualization requests with a predetermined time constraint.* We focus on middleware-based solutions, with the advantage that they treat the backend database as a black box without changes, and can leverage the computing capabilities to do in-situ analytics. We consider both rewritings that return exact results and rewritings that return approximate results. As a motivating example, consider a system that visualizes social media tweets on the US map with a time constraint of 500*ms*. Its backend database has a `tweets` table with attributes `Content`, `Location`, and `CreateAt`.

**Equivalent rewriting options.** Suppose a user asks for a spatial heatmap of tweets containing the keyword `covid` on the Thanksgiving day of 2020 in a region. The middleware creates a SQL query shown in Figure 1(a), which takes 3.35 seconds to execute. For this query, the physical plan generated by the database uses the keyword to access the inverted index on the `Content` attribute to retrieve candidate records, then filters them using the other two conditions. If we rewrite the query to an equivalent query by adding a hint (Figure 1(b)), the rewritten query takes only 0.3 seconds, as the hint helps the database generate a more efficient physical plan that uses the temporal filtering condition to access the B+ Tree index on the `CreateAt` attribute.




**(a)** The original SQL query takes 3.35*s*.

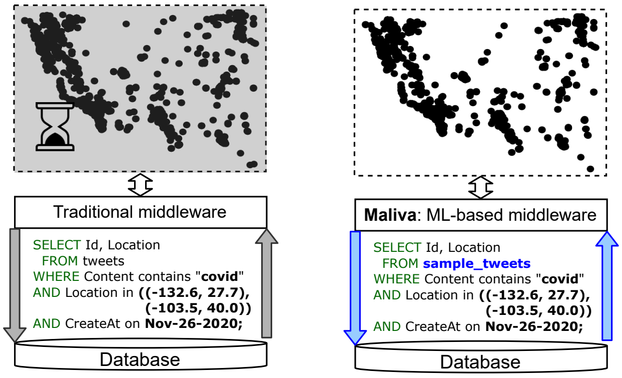
**(b)** A rewritten query with a hint takes 0.33*s*.

**Figure 1:** Equivalent rewriting option: adding query hints helps the database compute results within a time budget (500*ms*).

**Approximation rewriting options.** Figure 2(a) shows another visualization request on a larger region, which takes at least 4.28*s* for the database to run, no matter what hints we add. In this case, we rewrite the query by using random sampling, resulting in an approximation query that takes only 0.45*s* to run (see Figure 2(b)).

**Why does the database fail?** For the query in Figure 1(a), there are many reasons the database can fail to generate an efficient plan. One is the estimation error of the query cost due to an underestimation of the keyword `covid`'s selectivity. The cost-estimation problem in optimizers is notoriously hard [31]. For example, in our experiments (Section 7), out of the 602 visualization queries that had at least one physical plan that could finish within 500*ms*, PostgreSQL failed to choose an efficient plan for 269 queries due to its cost-estimation errors. Although there are many higher-accuracy estimators such as [17, 35, 37, 47, 56, 67, 68], their higher estimation cost prevents them from being adopted by a general-purpose database to meet the visualization need. In particular, for OLTP queries that need to be finished within milliseconds, spending tens of milliseconds for the cost estimation is unacceptable. A key observation is that for visualization applications where requests come with a time constraint, the

© 2023 Copyright held by the owner/author(s). Published in Proceedings of the 26th International Conference on Extending Database Technology (EDBT), 28th March-31st March, 2023, ISBN 978-3-89318-088-2 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.




**(a)** The query takes 4.28$s$ (no hints can reduce it).

**(b)** A rewritten query using a sample table takes 0.45$s$.

**Figure 2:** Approximation rewriting option: rewriting the query to compute an approximate result within the time constraint.

middleware can afford to spend more time (e.g., 300ms) on the cost-estimation using the high-accuracy estimators to find efficient plans (e.g., within 50ms), while it can still answer requests within a given time constraint (e.g., 500ms).

**Challenges.** We may enumerate all possible rewritten queries by applying different hints to a given query. We then use one of the aforementioned query-time estimators ("QTE" for short) to estimate the execution time of these rewritten queries and choose the most efficient one. There are several challenges in using this approach in the context of interactive visualization. **(C1)** A main challenge is that the cost of estimating the execution time of a rewritten query can be significant given a tight time constraint. For example, in Bao [35], estimating the execution time of all rewritten queries for one original query can take up to 230$ms$ in their experiments. **(C2)** Another challenge is the uncertainty caused by the estimation error of the QTE, and the fact that the backend database may or may not follow the provided hints to generate a physical plan. **(C3)** The third challenge is quality. For queries without equivalent rewritten queries that can meet the time constraint, approximate rewriting options need to be explored. It is critical to maximize the quality of the result while ensuring the query time is within the time constraint.

We address these challenges by introducing a novel machine-learning-based technique called Maliva, which stands for "Machine Learning for Interactive Visualization." The technique formulates the middleware task as a Markov Decision Process (MDP). For a given time budget, we train an MDP agent to balance the planning time and the execution time of the rewritten queries. By learning from previous experiences, the MDP agent judiciously explores different rewriting options, so that the total time (including planning and query execution) is within the time limit. (We address challenge **C1** in Section 4.) Using reinforcement learning to train the models, Maliva can handle the uncertainties introduced by the inaccurate time estimation and the fact that the database could ignore the query hints. (We address challenge **C2** in Section 5.) By considering visualization qualities of rewritten queries in the reward design of the MDP model, Maliva makes the best effort to maximize the result's quality while ensuring the query time is within the time limit. (We address challenge **C3** in Section 6.) Our experiments show that Maliva has a much higher chance (70×) than the original query to generate an execution plan such that the total time is within a time limit. Interestingly, it can also reduce query execution time. Both improvements show the significant benefits of adding learning capabilities to the middleware to support responsive visualization.

The rest of the paper is organized as follows. After formulating the middleware query-generation problem in Section 2, we give an overview of Maliva in Section 3. We present the details of this MDP-based solution, including its states, actions, transitions, and rewards (Section 4). We present how Maliva trains an MDP agent offline and uses it to generate a rewritten query online (Section 5). We generalize the MDP-based solution to be quality-aware by considering approximation rewriting (Section 6). Lastly, we report the results of a thorough experimental evaluation of Maliva to show its performance and benefits (Section 7).

## 1.1 Related Work

Visualization is a broad topic studied in many communities, and here we focus on efficiency-related works. A survey [15] summarized studies on interactive data analytics and visualization, and there are several recent studies on this topic [22, 24, 26, 50].

*Approximate Query Processing (AQP).* There are many techniques for computing approximate answers to queries [7, 12, 16, 27, 41, 45, 46, 48, 52, 62, 72, 73]. These approaches focus on developing approximation solutions to compute high-quality visualization. Existing solutions can be adopted as approximation rules in Maliva, such as Sample+Seek [12], which generates error-bounded visualization results by running queries on a small sample table.

*Datacube-based approaches.* Related studies include [10, 11, 21, 23, 28, 30, 40, 65]. In these approaches, the predefined cube intervals cannot support visualization queries with arbitrary numerical range conditions. The proposed Maliva system efficiently computes results for visualization queries with arbitrary conjunctive selection conditions.

*Progressive visualization.* There are solutions to show visualization results progressively [7, 9, 10, 14, 20, 39]. For instance, DICE [10] uses random and stratified samples to present an approximate result and then incrementally updates the result. These progressive visualization systems can adopt the proposed Maliva middleware to further optimize the intermediate queries to increase their efficiency.

*Prefetching-based approaches.* Techniques including [1, 5, 53, 69] accelerate visualization queries by prefetching or caching their results. For example, ForeCache [5] divides visualizations into tiles and prefetches them based on predicted user behaviors. Maliva is orthogonal to these techniques, and it can be adopted by them to further optimize the database queries.

*Visualization using big data systems.* These techniques use Hadoop, Spark, and Hive to support visualizations [7, 8, 13, 58, 70]. For instance, HadoopViz [13] and GeoSparkViz [70] use Hadoop and Spark to generate high-resolution visualizations. Their focus is on offline construction, not on an interactive visualization for queries with ad-hoc conditions. The proposed Maliva middleware technique is complementary to these solutions.

*ML for visualization.* A survey by Wang et al. [63] summarized studies of applying ML techniques to different stages during the whole visualization pipeline. Examples are [32, 64] for data cleaning and preparation and [19, 33, 51] for visualization recommendation. Our proposed system focuses on applying ML techniques to solve performance issues at the middleware.

*ML-based query optimization.* ML has recently used in database optimizers [25, 35–37, 54, 61, 71], selectivity estimation [17, 47], and cost estimation [56]. **Comparison with Bao**: The recent

Bao technique [35] uses hints to generate optimized queries by modeling the optimization as a multi-armed bandit problem. It applies Thompson sampling to minimize the training time and maximize the accuracy of its neural-network-based query time estimator (QTE). We have a detailed discussion about the differences between Maliva and Bao in Section 6.3. We also conducted extensive experiments in Section 7 to compare their performance, and the results show Maliva outperformed Bao in various metrics (See Section 7.6).

## 2 PROBLEM FORMULATION

**Visualization architecture.** We consider a typical three-tier data-visualization system that consists of a backend database, a middleware layer, and a frontend. For each frontend visualization request, let $Q$ be the *original* SQL query for the request. Let $\tau$ be a time limit that quantifies the expected responsiveness of the system. Ideally, we want the total delay, from the time the user submits a request to the time the result is shown on the frontend, to be within $\tau$. The original query $Q$ may not meet the time-limit constraint when the backend database cannot generate a physical plan that is fast enough. To solve this problem, Maliva rewrites $Q$ with two kinds of options: *query hints* and *approximation rules*. By adding a query hint to $Q$, Maliva can help the backend database generate an efficient physical plan that computes the result within the time limit. For expensive queries where no physical plan can meet the time limit, Maliva can add an approximation rule to the original query such that the backend database computes an approximate result to trade the visualization quality for responsiveness. Note that the proposed approach also works in a more general setting of approximate query processing (AQP) where a time constraint is given.

**Query hints.** A *query hint* in a database is an addition to the SQL standard that instructs the database engine on how to execute the query. For example, a hint may tell the engine to use or not to use an index (even if the query optimizer would decide otherwise) [18]. A query hint does not change the semantic meaning of the query, i.e., the result computed by the database engine with the hint remains the same. Databases such as AsterixDB [2], MySQL [42], Oracle [44], PostgreSQL [49], and SQL Server [55] support a variety of query hints. For example, in Figure 3(b), Maliva adds two hints *+ Index-scan(t CreateAt)* and *Nest-Loop-Join(t u)* to the original query. They suggest the engine to use the index on the `CreateAt` attribute to scan the table `t`, and do a nest-loop join on tables `t` and `u`.

(a) Original Query (Q)

```
SELECT BIN_ID, COUNT(*)
  FROM tweets t, users u
WHERE t.Content contains "covid"
    AND t.Location in ((-124.4, 32.5),
                       (-114.1, 42.0))
    AND t.CreateAt on 'Nov-26-2020'
    AND u.TweetCnt in [100, 5000]
    AND t.user_id = u.id
  GROUP BY BIN_ID(t.Location);
```

(b) Rewritten Query (RQ)

```
/*+ Index-scan(t CreateAt),
    Nest-Loop-Join(t u) */
SELECT BIN_ID, COUNT(*)
  FROM tweetsSample20 t, users u
WHERE t.Content contains "covid"
    AND t.Location in ((-124.4, 32.5),
                       (-114.1, 42.0))
    AND t.CreateAt on 'Nov-26-2020'
    AND u.TweetCnt in [100, 5000]
    AND t.user_id = u.id
  GROUP BY BIN_ID(t.Location);
```

**Figure 3:** A original query and a rewritten query.

**Approximation rules.** An *approximation rule* is a method to rewrite the original SQL query to compute an approximate result, and the new query takes less time. There are various approximation rules available in database systems, such as adding a *"Limit"* clause, applying a SQL-standard *"TableSample"* operator on a table, or substituting a table with a smaller table randomly sampled from the original table. For example, in Figure 3(b), Maliva rewrites the original query by substituting the table `tweets` with a sample table `tweetsSample20` with 20% randomly selected records.

Now we formally define rewriting options, rewritten queries, and the query-rewriting problem.

*Definition 2.1.* (Rewriting Option) Let $H$ be a set of query-hint sets and $A$ be a set of approximation-rule sets. A rewriting option ("RO" for short) is a tuple $(h, a)$, where $h \in H$ and $a \in A$. Note that both $h$ and $a$ can be the empty set $\emptyset$.

For instance, the rewriting option in Figure 3(b) is a tuple with a query-hint set of "use the index on `CreateAt` and do a nest-loop join on `t` and `u`" and an approximation-rule set of "substituting the table `tweets` with the sample table `tweetsSample20`". We assume the user-defined candidate set of rewriting options does not contain invalid query-hint sets or approximation rules. A query-hint set is considered to be invalid if it contains conflicting hints, e.g., *Nest-Loop-Join(t u)* and *Hash-Join(t u)*.

*Definition 2.2.* (Rewritten Query) Given an original SQL query $Q$ and a rewriting option $RO$, a rewritten query ("RQ" for short) is a new SQL query generated by applying $RO$ onto $Q$. If $RO = (\emptyset, \emptyset)$, then $RQ = Q$.

For example, Figure 3(b) is a rewritten query for the original query in Figure 3(a).

**Query-rewriting problem.** Given a visualization request's original SQL query $Q$, and a time limit $\tau$, we want to generate a rewriting option, such that the total time of the corresponding rewritten $RQ$, including planning and query execution, is within $\tau$ and the quality of $RQ$'s result is maximized. To quantify the quality, we assume a given visualization quality function $F$. Let $r(Q)$ be the result of the original query $Q$ and $r(RQ)$ be the result of the rewritten query $RQ$. Then $F(r(Q), r(RQ))$ computes the quality of $r(RQ)$.

In Sections 3, 4, and 5, we study the case of using query hints only (i.e., without changing query results). In Section 6, we study the case where approximation rules are also used.

## 3 MALIVA: ML-BASED QUERY REWRITING

We now introduce the middleware technique called "Maliva" to solve the aforementioned query-rewriting problem. We first give an overview of the technique, then use an example to explain the details.

**Overview.** As illustrated in Figure 5, Maliva rewrites the original SQL query to answer a visualization request within a time budget. It considers a predefined set of rewriting options, which we denote as $\Omega = \{RO_1, \ldots\}$. For each $RO_i$, the rewritten query is denoted as $RQ_i$. The set of candidate rewritten queries is $\Phi = \{RQ_1, \ldots\}$.

Maliva has a *Query Rewriter* that enumerates possible RQs and uses a *Query Time Estimator* (QTE) to estimate the execution time of each of them. The *Query Rewriter* uses the best effort to choose an RQ such that the total time, including the planning process and query execution, is within the time budget $\tau$. Such an RQ is called *viable*. The middleware then sends the rewritten query to the database. The *Query Result Handler* sends the retrieved result to the frontend to visualize.

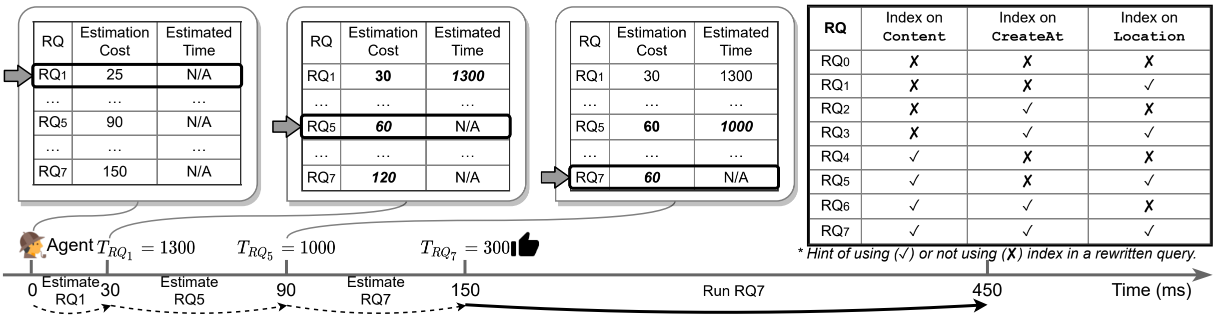



**Figure 4:** The Query Rewriter acts like an agent who makes a sequence of decisions to generate a rewritten query (with a total time $\leq 500ms$). At time 0, the agent considers rewritten query $RQ_1$ due to its low estimation cost (estimated $25ms$, the actual $30ms$ is updated once $RQ_1$ is explored). After estimating its execution time ($1,300ms$), the agent knows that $RQ_1$ is not viable since the total time is longer than $500ms$. The estimation of $RQ_1$ affects the costs for estimating $RQ_5$ and $RQ_7$. The agent explores $RQ_5$ and then $RQ_7$. With the estimated execution time being $300ms$ and the elapsed time being $150ms$, $RQ_7$ is decided as a viable rewritten query because the total time ($450ms$) is within $500ms$.

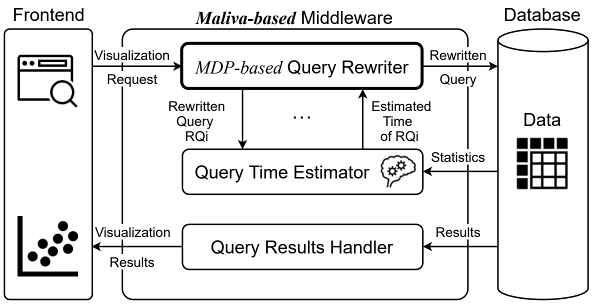



**Figure 5:** Overview of Maliva.

Naïvely enumerating all available RQs in $\Phi$ is computationally prohibitive due to two reasons. First, the cost of *Query Time Estimator* to estimate the execution time of a rewritten query is not negligible. For instance, in some cases it could take up to $70ms$ [56] on a $7GB$ dataset or $300ms$ [67] on a $10GB$ dataset. Second, the number of RQs increases exponentially when the number of applicable indexing choices increases. For example, consider a selection query on a table with filtering conditions on $m$ attributes, and the database has an index on each attribute. The number of query-hint sets in $H$ would be $2^m$, since the database can use any subset of the $m$ indexes to do filtering and then intersect the record lists to compute the final result. Therefore, the *Query Rewriter* needs to balance the exploration time for query estimation and the execution time of each chosen RQ to find a viable RQ.

**An example.** Maliva views query rewriting as a Markov decision process (MDP) [57] and adopts machine learning (ML) to solve this problem. We use the running example in Section 1 to illustrate how Maliva uses an MDP agent to make a sequence of decisions to find a viable RQ. For simplicity, we assume the rewrite-options (RO) set to involve query hints only. We will generalize the technique to consider approximation rules in Section 6. As shown in Figure 4, a query has three selection conditions on three attributes, and each of which has an index. Suppose in the set $H$ of query-hint sets, each attribute has a query hint of using or not using the index. Thus, we have $2^3 = 8$ query-hint sets to choose from. The agent makes a sequence of decisions to estimate the execution times of several rewritten queries and find a viable one $RQ_7$ (that uses the indexes on all three attributes). Next, we present the details of this MDP-based technique.

# 4 MDP MODEL FOR ADDING QUERY HINTS

In this section, we present the details of using an MDP model in Maliva to solve the query-rewriting problem and discuss how to implement the Query Time Estimator (QTE).

## 4.1 MDP Model for Query Rewriting

MDP [57] is a formalization of sequential decision-making problems where an agent learns to achieve a goal from interaction with an environment. At each time step, the agent is in a state $s$, and chooses an action $a$ available in state $s$. The environment transits the agent to a new state $s'$, and gives the agent a corresponding reward $R(s, a)$. To train an MDP agent is to find a good policy $\pi_*$ such that if the agent follows the policy to choose an action for each state, it maximizes the total reward.

We use the MDP model to solve the query-rewriting problem. For simplicity, we first focus on the case where rewriting options do not contain any approximation rules, which means no rewritten queries have quality loss. We will generalize the technique to consider approximation rules in Section 6. Without considering quality loss, the MDP agent learns to maximize the chance of finding a viable rewritten query for a given visualization request. The agent takes a sequence of actions, and each action chooses an RQ to explore. That is, it asks the query time estimator (QTE) to estimate the execution time of the RQ. The agent chooses an RQ based on the current state, and considers the future cost it needs to pay and the execution time of RQs already explored. The agent gets a bonus if it finds a viable RQ, or a penalty if it runs out of time. In the offline phase, by analyzing queries in the training workload, the agent learns to maximize the chance to receive a bonus. In the online processing phase, given a new query, the agent decides which RQ to explore in each step to receive a bonus in the end. Now we present the details of how to use MDP to model the process of choosing RQs.

**States.** A state represents the past decisions, based on which the agent decides an RQ to consider next. Suppose we are given a predefined set of $n$ ROs, i.e., $\Omega = \{RO_1, \ldots, RO_n\}$. Correspondingly, we have $n$ candidate RQs, denoted as $\Phi = \{RQ_1, \ldots, RQ_n\}$. A state is a vector

$$s = (E, C_1, C_2, \ldots, C_n, T_1, T_2, \ldots, T_n),$$

which includes three pieces of information, as shown in Figure 6. (1) The elapsed time ($E$) captures how much time we have spent. (2) The estimation cost ($C_i$) for each possible rewritten query $RQ_i$ captures how much time is needed for the agent to estimate its running time. Each $C_i$ is initialized with a rough estimation collected offline and updated during the online planning phase. Note that the MDP state does not require the initial $C_i$ values to be accurate, and a rough estimation from history statistics suffices. The actual estimation costs will be collected while the MDP agent processes a query, as will be described soon in the definition of *Transitions.* (3) $T_i$ is the estimated time for each already explored $RQ_i$. Each $T_i$ is initialized with a 0 value until it is filled with an estimated execution time.

| | Elapsed time | Estimation cost $C_i$ | | | | Estimated time $T_i$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $RQ_1$ | $RQ_2$ | ... | $RQ_n$ | $RQ_1$ | $RQ_2$ | ... | $RQ_n$ |
| State $s = ($ | $E$ | $C_1$ | $C_2$ | ... | $C_n$ | $T_1$ | $T_2$ | ... | $T_n$ ) |

**Figure 6:** An MDP state in Maliva.

We assume for each rewritten query $RQ_i$, collecting the physical plan and its statistics (e.g., cardinality and cost estimations of each operator) is done by the QTE, and its time is captured by the MDP model's estimation cost ($C_i$). We assume the rewritten queries' physical plans and statistics are not available to the MDP model. Thus, the proposed MDP model is general, and can be applied to any query shape with any predefined query-hint set. A natural question is that without the statistics of the explored RQs stored in the state, how can the MDP model make a good decision on which RQ to choose next? Our answer is that the execution time of a rewritten query implicitly captures the statistics of the physical operators (e.g., the cost of doing an R-Tree index scan on the `Location` attribute). By keeping the estimated execution time of each explored RQ in its state, the MDP model can learn the correlations of the execution times between different rewritten queries and make good decisions.

**Actions.** An action, denoted as $a$, is to explore an RQ next. For each RQ, the agent asks the QTE to estimate its execution time. Meanwhile, the agent needs to pay a cost as it takes time for the QTE to extract query features, possibly by collecting online statistics from the database, and running the estimation model to do the estimation. In the running example, at time 0, the agent decides to explore $RQ_1$. It asks the QTE to estimate $RQ_1$'s execution time.

**Transitions.** A transition function defines how the environment computes the next state, given the agent's action in the current state. Let the RQ considered by action $a$ in state $s$ be $RQ_i$. We define the transition function $\mathcal{T}$ as follows. First, the QTE estimates the time of $RQ_i$, and we add the estimated time $T_i$ to the state. Second, the estimation costs for other RQs could change. In the running example, to estimate $RQ_1$ that uses the R-Tree index on the `Location` attribute, we need to collect the spatial filtering condition's selectivity on the `Location` attribute. To estimate $RQ_5$ that uses both the inverted index on the `Content` attribute and the R-Tree index on the Location attribute, we need to collect the selectivity values of the filtering conditions on both attributes. After the agent takes the $RQ_1$ action, we update the estimation cost of $RQ_1$ to be the actual time it costs and update the estimated estimation cost of $RQ_5$ by excluding the cost to collect the selectivity value of the spatial filtering condition. As shown in Figure 7, after estimating the time of $RQ_1$, we add the estimated time $1,300ms$ of $RQ_1$ to the state, update the estimation cost for $RQ_1$ from the estimated $25ms$ to the actual $30ms$, and update the estimation cost for $RQ_5$ from the previous estimated $90ms$ to the new estimated $60ms$. Lastly, the estimation takes time $\hat{C}_i$, and we add it to the elapsed time so far to indicate how much time the agent has spent exploring different RQs. Note that the $\hat{C}_i$ is the actual cost of estimating $RQ_i$, which could be different from $C_i$ because $C_i$ is an estimated cost for estimating $RQ_i$.

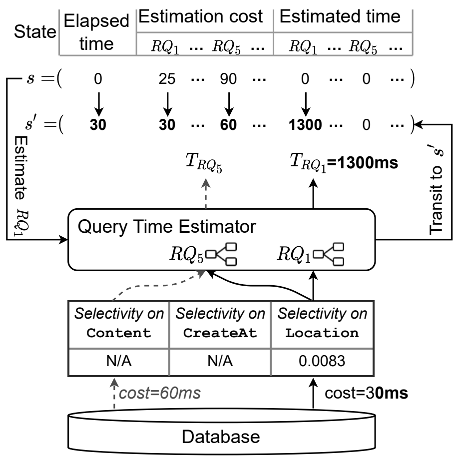



**Figure 7:** Transition after estimating execution time of $RQ_1$.

**Rewards.** A reward function defines the agent's immediate gain when it takes a particular action $a$ in a given state $s$. In our setting, consider two cases to compute the reward function. (1) The first case is when the agent is at an intermediate state where it still has time for planning but has not yet found a viable rewritten query. In this case, the agent should not be awarded or punished since it has not made a decision yet. Thus the reward value is 0. (2) The second case is when the agent is at a termination state where it decides the rewritten query $\hat{RQ}$, runs it against the database, and collects the execution time $\hat{T}$ of $\hat{RQ}$. In this case, the agent should be awarded if the total time (including both the planning time and rewritten query execution time) is less than the time budget, or punished if the total time is more than the budget.

The agent decides on a rewritten query by considering three situations. The first one is that the agent finds an RQ to be viable based on the estimation of the QTE before running out of time. For example, in Figure 4, after spending $150ms$ for planning, the agent decides $RQ_7$ as the chosen rewritten query, since the predicted total time of $RQ_7$ is $450ms$, which is within the $500ms$ budget. The second situation is when the agent uses up the time budget and has to stop planning. The third situation is when the agent has exhausted all candidate rewritten queries and has to decide which RQ to choose. In the latter two situations, the agent chooses the fastest RQ known so far as the final decision.

Formally, suppose the generated rewritten query by the agent is $\hat{RQ}$ and the actual running time of query $\hat{RQ}$ is $\hat{T}$. Then the reward function $\mathcal{R}(s, a)$ is defined as follows,

$$\mathcal{R}(s, a) = \frac{(\tau - s.E - \hat{T})}{\tau}, \qquad (1)$$

where $s.E$ denotes the elapsed time so far in state $s$. If the total time $s.E + \hat{T}$ is less than the time budget $\tau$, which makes $\mathcal{R}(s, a)$ positive, then the agent receives a reward. The faster the rewritten query is, the larger the reward will be. On the other hand, if the total time exceeds the time budget, which makes $\mathcal{R}(s, a)$ negative, then the agent receives a penalty. The slower the rewritten query is, the larger the penalty will be. Thus, guided by the reward function, the MDP model will learn to find an efficient rewritten query as soon as possible.

### 4.2 Query Time Estimator (QTE)

Take the sampling-based QTE described in [67] as an example. It first builds an analytical cost model (e.g., linear regression model), and uses it to estimate the execution time of a rewritten query by collecting its statistics online. Specifically, it estimates the selectivity values of the query conditions by running `count(*)` queries on a small sample table, provides the values as input features to the cost model, and uses the model's prediction as the query's execution-time estimation. There are also other possible solutions in the literature [37, 56, 68] that can be used by Maliva. Note that QTEs are the focus of this paper, and Maliva leverages a given QTE intelligently to balance the planning time and the query execution time.

## 5 TRAINING AND USING THE MDP AGENT

In this section, we discuss how to train the MDP agent offline in Maliva on a workload of visualization requests and use it to generate a viable rewritten query online.

### 5.1 Training the MDP Agent

Suppose we have a workload of queries $W = [q_1, q_2, \ldots, q_m]$. Our goal is to find an optimal policy $\pi^*$ such that for any query $q_i \in W$, the agent following policy $\pi^*$ maximizes the chance to generate a viable rewritten query. We adopt the *deep Q-learning* algorithm [38] for finding an optimal policy for the MDP agent. Its main idea is to use a neural network (called *Q-network*) to represent a policy $\pi$. Given an input of a state vector, the q-network outputs a *Q-value* [66] for each possible RQ in the state. A higher q-value means that the rewritten query is more likely to be viable given the current information. More details of the Q-network design can be found in the full version of this paper [3]. Its training process includes two main steps. The first step is to generate a set of experiences by exploring different planning sequences for queries in the workload repeatedly. The second is to replay those experiences to update the q-network's weights gradually such that the q-network can approximate the q-values of the optimal policy for each state-action pair.

**Training an MDP agent for query rewriting.** Algorithm 1 details how we train the MDP agent. To apply deep q-learning, we generate the replay memory $M$ of experiences. For a given visualization query workload $W = [q_1, q_2, \ldots, q_m]$, we generate a set of experiences. Each experience is a 4-tuple

$$(s, a, s', r'),$$

where the agent in a state $s$ estimates the time of the hinted query represented by an action $a$ and observes the next state $s'$ with a reward $r'$. Note that different queries can have different optimal policies. Our goal is to learn an optimal policy for the whole workload. We let the agent explore all the queries in the workload $W$ in multiple iterations until the policy converges or the number of runs exceeds a maximum threshold. In each iteration, we shuffle the order of queries to reduce the bias caused by earlier queries on the exploration direction of later queries. For each query $q$ in $W$, we let the agent complete a sequence of decisions. At each step, it selects an RQ to estimate. It pays the cost to estimate the rewritten query's execution time, transits to the next state, and receives an immediate reward. The agent repeats the process until it reaches a termination state (line 9) in one of the three cases. The first case is when the estimated time $T(a)$ of the rewritten query in action $a$ suggests it is potentially viable, i.e., $s.E + T(a) \leq \tau$. The second case is when the agent runs out of time, i.e, $s.E \geq \tau$. The third case is when the agent has exhausted all possible RQs, i.e., $\rho = \emptyset$.

**Algorithm 1:** Training an MDP agent

**Input:** A query workload $W = [q_1, q_2, \ldots, q_m]$
A transition function $\mathcal{T}$
A reward function $\mathcal{R}$
A time budget $\tau$
**Output:** An agent's policy $\pi$

1 Replay memory $M \leftarrow \{\}$ with capacity $C$;
2 Initialize policy $\pi$ with random weights;
3 **while** *$\pi$ does not converge* **do**
4   $W \leftarrow$ shuffle the queries in $W$;
5   **for** *each query $q$ in $W$* **do**
6     State $s \leftarrow (0, C_1, C_2, \ldots, C_n, 0, 0, \ldots, 0)$;
7     Remaining set $\rho \leftarrow$ query $q$'s all possible RQs $\{RQ_1, RQ_2, \ldots, RQ_n\}$;
8     Reward $r \leftarrow 0$;
9     **while** *$(s, \tau, \rho)$ is not at a termination state* **do**
10       $f \leftarrow$ generate a random number from [0,1];
11       **if** $f < \epsilon$ **then**
12         $a \leftarrow$ a random RQ from $\rho$;
13       **else**
14         $a \leftarrow \arg\max_{RQ_i \in \rho} Q^{\pi}(s, RQ_i)$;
15       **end**
      *// Estimate query $a$ and transit to state $s'$*
16       $s' \leftarrow \mathcal{T}(s, a)$;
      *// Compute the immediate reward*
17       $r' \leftarrow \mathcal{R}(s, a)$;
18       Store experience tuple $(s, a, s', r')$ in $M$;
      *// Remove query $a$ from the remaining set $\rho$*
19       $\rho \leftarrow \rho - \{a\}$; $s \leftarrow s'$; $r \leftarrow r'$;
20     **end**
21     Update $\pi$ using a random sample from $M$;
22   **end**
23 **end**

When the agent decides which RQ to explore at each step (lines 12 to 14), we adopt an $\epsilon$-greedy strategy [38, 59] to balance between the exploration of RQs with uncertain values and the exploitation of RQs known with high values. With an $\epsilon$ probability, the agent chooses a random RQ that has not been considered before (line 12). Otherwise, it selects an RQ that has not been explored with the highest q-value based on the current policy weights (line 14). We start with a high probability ($\epsilon$) of exploration and gradually decrease it to favor exploitation with the training progress.

Once an RQ is decided by the agent as an action $a$, we call the transition function $\mathcal{T}$ (Section 4.1) to transit the agent to the next state $s'$ (line 16). We estimate the query $a$'s running time and update the new state $s'$ by adding the estimated time for $a$, adding the cost to the elapsed time, and modifying the costs of affected RQs. We then call the reward function $\mathcal{R}$ (Section 4.1) to compute an immediate reward $r'$ for estimating the RQ in $a$ (line 17). To this end, we have generated a new experience tuple $(s, a, s', r')$, and store it in the replay memory $M$ (line 18). When $M$ reaches its capacity $C$, we replace existing experiences in a FIFO manner.

After processing a query, we update the policy $\pi$ following the original deep q-learning algorithm [38] (line 21). We sample a random subset $M'$ of experiences from $M$. For each experience tuple $(s, a, s', r')$ in $M'$, we first compute the target q-value $y$ of the state-action pair $(s, a)$ using the Bellman equation [66]. We then update the weights in policy $\pi$ by minimizing the loss value $L$ between the target q-value $y$ and the current q-value, where $L$ is defined as:

$$L = (Q^{\pi}(s, a) - y)^2.$$

We keep updating the policy $\pi$ until it converges, i.e., the total accumulated reward of the training workload $w$ does not improve much in new iterations (e.g., less than 1%).

**Accommodating estimation inaccuracy using MDP.** One advantage of using the MDP framework where an approximate QTE may give inaccurate estimations is its tolerance of the inaccuracy. The MDP model captures the uncertainty in two places. One is the transitions between states that store the estimated times of explored RQs. Although estimated times can have errors, statistically, after learning from the historical queries, the agent understands which action has the highest expected total reward. Another place is the reward definition, where the penalty for making a wrong decision will lead the agent to understand the QTE's mistakes and avoid them in the future.

### 5.2 Using MDP to Rewrite Queries Online

After we train an MDP agent, the query rewriter utilizes the agent to generate a rewritten query for a new visualization query $q$ online. Algorithm 2 shows the pseudo-code. Starting from an initial state $s$, we use the trained policy $\pi$ to compute the q-values for all the RQs and select the one with the highest q-value as the action $a$ (line 5). We then estimate the running time of query $a$ and transit to state $s'$ (line 6). We compute the immediate reward $r'$ for estimating RQ in $a$ (line 7). If the action $a$ is a potentially-viable RQ (line 9), we output the query $\hat{RQ_i}$ in $a$ as the generated rewritten query. Otherwise, we run out of time for the remaining RQs (line 11). Then we select the rewritten query $RQ_j$ with the minimum execution time estimated so far and output it. If neither cases happen, we repeat the above process.

**Algorithm 2:** Generating an RQ online

**Input:** A new query $q$
A trained policy $\pi$
A transition function $\mathcal{T}$
A reward function $\mathcal{R}$
A time budget $\tau$
**Output:** An RQ

1 State $s \leftarrow (0, C_1, C_2, \ldots, C_n, 0, 0, \ldots, 0)$;
2 Remaining set $\rho \leftarrow$ query $q$'s all possible RQs $\{RQ_1, RQ_2, \ldots, RQ_n\}$ ;
3 Reward $r \leftarrow 0$;
4 **while** *True* **do**
  *// Select a query with the highest q-value predicted by $\pi$*
5   $a \leftarrow \arg\max_{RQ_i \in \rho} Q^{\pi}(s, RQ_i)$;
  *// Estimate query $a$ and transit to state $s'$*
6   $s' \leftarrow \mathcal{T}(s, a)$;
  *// Compute the immediate reward*
7   $r' \leftarrow \mathcal{R}(s, a)$;
  *// Remove query $a$ from the remaining set $\rho$*
8   $\rho \leftarrow \rho - \{a\}; \leftarrow s'; r \leftarrow r'$;
9   **if** $s.E + T(a) \leq \tau$ **then**
10    **return** $\hat{RQ_i}$ represented by $a$;
11   **if** $s.E \geq \tau$ *or* $\rho = \emptyset$ **then**
12    **return** $RQ_j$ with the minimum execution time estimated in $s$;
13 **end**

## 6 APPROXIMATION REWRITING OPTIONS

In this section, we generalize Maliva by considering rewriting options with approximation rules. Recall that using a query-hint set to rewrite an original query $Q$ into an $RQ$ can help the database generate an efficient physical plan that computes the actual result without any approximation. However, for expensive queries where no physical plan can meet the time constraint, by applying an approximation-rule set to $Q$, Maliva can generate an $RQ$ that efficiently computes an approximate result within the time budget. We first extend the MDP model in Section 4 to consider approximation rules. We then discuss two approaches to applying the MDP model to implement a quality-aware query rewriter. The quality-aware query rewriter makes the best effort to generate a viable rewritten query and maximize the result's quality. In the end, we discuss the trade-offs between the two approaches.

### 6.1 Quality-Aware MDP Model

Consider the case where the rewriting options contain both query hints and approximation rules. A rewritten query can return an approximate result with quality loss. We need to let the MDP agent learn to maximize the chance to generate a viable rewritten query and maximize the quality of the query result simultaneously. To quantify the quality of a rewritten query, we assume a given visualization quality function $F$. Let $r(Q)$ be the result of the original query $Q$, and $r(RQ)$ be the result of the rewritten query $RQ$. Then $F(r(Q), r(RQ))$ computes the quality of $r(RQ)$. For example, suppose we use the Jaccard similarity function to measure the quality of an approximate result. Figure 8 shows that the quality of the scatterplot visualization result of an approximate rewritten query $RQ$ compared to the original query $Q$ is 0.76. Note that Maliva does not have restrictions on quality functions, and many functions can be used, such as VAS in [45] for scatterplots and the function of distribution precision in [12] for pie charts.

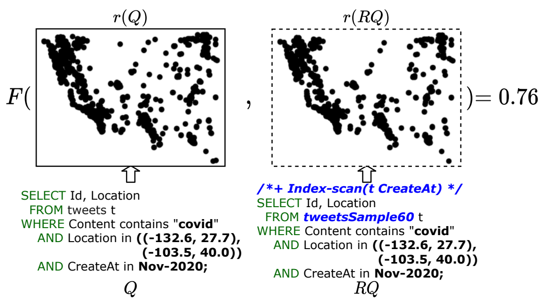



**Figure 8:** The quality of $RQ$ compared to $Q$ using a Jaccard-based quality function as an example.

**Reward function for a quality-aware MDP model.** To achieve the goal of guiding the MDP agent to learn to maximize the chance to generate a viable rewritten query and maximize the quality of the query result simultaneously, we extend the definition of the reward function in Section 4. Recall that the learning goal of an MDP agent is to maximize the accumulative reward. In Section 4, once the agent decides on a rewritten query, it receives a reward that reflects the query performance in terms of the total running time. Guided by the reward, the agent learns to generate a viable rewritten query quickly. Similarly, the MDP agent can also learn to quickly generate a viable rewritten query with a high result quality if the final reward reflects both the decided rewritten query's efficiency and quality. The main idea is to combine the efficiency defined in Section 4 and the quality. The new reward function is a weighted summation of both. Formally, suppose the generated rewritten query by the agent is $\hat{RQ}$ and the actual running time of query $\hat{RQ}$ is $\hat{T}$. Then the new reward function $\mathcal{R}(s,a)$ is defined as follows:

$$\mathcal{R}(s,a) = \beta^{(\tau - s.E - \hat{T})/\tau} + (1-\beta)F\big(r(Q), r(\hat{RQ})\big). \qquad (2)$$

The term $(\tau - s.E - \hat{T})/\tau$ represents the efficiency of the rewritten query in terms of running time compared to the time budget. The function $F(r(Q), r(\hat{RQ}))$ represents the quality of the RQ's result. Note that computing $F$ could be expensive since the actual result $r(Q)$ of the original query is required. However, we only need to pay the cost in the offline training phase once. In the online phase, we don't need to compute the $F$ value for a new query when we use the MDP model to explore different RQs. Since the MDP model learns from the final reward values only, we do not require every query to use the same quality function. In particular, different quality functions can be applied for different training queries to evaluate their visualization qualities, e.g., some queries are visualized as scatterplots and others as heat-maps. $\beta \in [0,1]$ is a parameter that indicates how important the running time is compared to the result quality.

## 6.2 Quality-Aware Query Rewriter

Now we discuss how to apply the extended MDP model to implement a quality-aware query rewriter. We present the technical details of two approaches and discuss their pros and cons. We will show the evaluation results in Section 7.

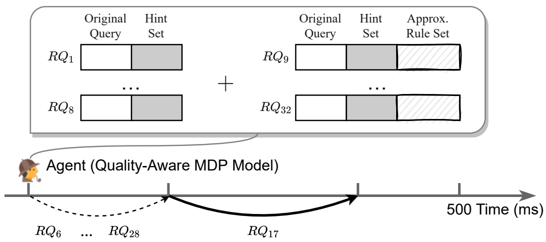



**Figure 9:** One-stage MDP approach.

**One-stage approach.** A natural idea is to replace the MDP model in Section 4 with the quality-aware MDP model. We let the MDP agent simultaneously consider query hints and approximation rules as rewriting options. By applying the new reward function combining both the efficiency of the rewritten query and the result's quality, the MDP agent learns to maximize the chance of generating viable rewritten queries and maximize the quality.

**Two-stage approach.** A drawback of the previous approach is that the agent might miss a non-approximate viable rewritten query. To solve this problem, we consider a two-stage approach, with a main idea to let the MDP agent exhaust all candidate query hints first and then explore those approximation rules. In the two-stage approach, Maliva first runs the original MDP model, excluding the approximation rules. If the agent finds a viable rewritten query, it outputs the $RQ$ as before. If the agent exhausts all candidate $RQ$s without finding a viable one, and the elapsed time has not exceeded the time budget $\tau$, then we run the new quality-aware MDP model that considers the approximation rules to find a viable $RQ$.

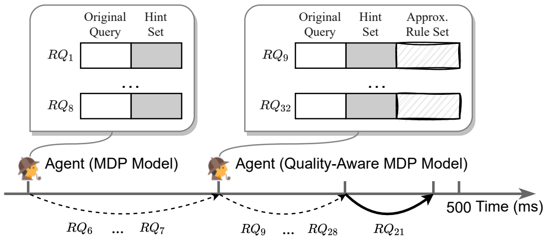



**Figure 10:** Two-stage MDP approach. After running the original agent that considers the 8 query-hint sets defined in Figure 4 without approximation rules, we cannot find a viable RQ. We then run the new agent with the quality-aware MDP model that considers all 8 query-hint sets combined with 3 approximation-rule sets (e.g., substituting the `tweets` table with 20%, 40%, or 80% sample tables), resulting in 24 rewritten queries in total. After spending extra time exploring a few RQs, the quality-aware agent chooses $RQ_{21}$ as the final decision.

When the planning time for the original agent is longer than the time budget, the two-stage approach reduces to the case described in Section 4. In this case, the one-stage approach is preferred since it can increase the chance of generating a viable rewritten query considering approximation rules. When the planning time for the original agent is relatively small compared to the time budget, the two-stage approach has the advantage of not missing any non-approximate viable rewritten queries.

## 6.3 Differences between Maliva and Bao

The recent Bao technique [35] also uses hints to rewrite queries. Maliva is closely related to Bao but different at multiple levels. First, to select a potentially viable query plan from all the candidate query-hint sets, Bao takes a brute-force approach by enumerating all options (using QTE). Maliva, in contrast, trains an MDP agent that explores the options by carefully balancing the planning time and query execution time. This difference makes Bao's method inapplicable in many visualization problems in the main application domain of Maliva. Secondly, as we will demonstrate in Section 7.6, when the number of candidate rewriting options is large (e.g., > 16), the planning time of Bao can exceed the time budget. Maliva, on the other hand, has a significantly shorter planning time and thus is capable of generating much more viable rewritten queries. Lastly, Bao does not consider approximate rewrites. Maliva, in contrast, offers flexibility by allowing approximate rewriting queries with minimal quality loss.

# 7 EXPERIMENTS

We conducted experiments to evaluate Maliva[1]. In particular, we want to answer the following questions: (1) How well does it rewrite queries to support visualization requests? (2) How well does it generalize to different numbers of rewriting options? (3) How well does it perform for different types of queries (e.g. single-table selection queries and multiple-table joining queries)? (4) How well does it generalize to different time budgets, unseen queries and other databases? (5) How does it compare with related solutions? and (6) How much is its training overhead?

[1]Maliva is open-sourced on Github (https://github.com/baiqiushi/maliva)

## 7.1 Setup

**Datasets.** We used two real datasets and a synthetic one as shown in Table 1. The Twitter dataset included 100 million geo-located tweets in the US from November 2015 to January 2017. We kept the timestamp, geo-coordinate, text message, and several user attributes for each tweet in a `tweets` table. For the experiment on join queries, we used the `tweets` table and a `users` table. The former had a foreign key of "user_id" referencing the "id" in the latter. We used the geo-coordinate attribute as the output for visualization (e.g., choropleth map, heatmap, or scatterplot). The NYC Taxi dataset [43] included taxi trip records within three years from 2010 to 2012. The third dataset was generated from the TPC-H benchmark [60]. We used the line-item table as the fact table. The attributes we used for query selection conditions are shown in Table 1.

**Table 1:** Datasets.

| Dataset | Record # | Size | Filtering Attributes |
|---|---|---|---|
| Twitter | 100,000,000 | 57GB | text, created_at, coordinates, users_statues_count, users_followers_count |
| NYC Taxi | 500,412,914 | 146GB | pickup_datetime, trip_distance, pickup_coordinates |
| TPC-H | 300,005,811 | 65GB | extended_price, ship_date, receipt_date |

**Query workloads.** We generated random queries on each dataset for training and evaluation. Take Twitter dataset as an example. We first randomly sampled a set of tweets from the base table. For each tweet, we generated a query as follows. We chose the `text`, `created_at`, and `coordinates` attributes for the selection conditions in the query. We generated three conditions based on the values in the sampled tweet. For `text`, we randomly selected a non-stop word in the original tweet's text message as the keyword condition. For `created_at`, we generated a temporal range condition with the value in the original tweet as the left boundary. We divided the maximum range on the `created_at` attribute in the base table into multiple zoom levels, and randomly selected a level to generate the length of the range condition. Suppose the maximum range on `created_at` had $L$ days. We computed the maximum zoom level on `created_at` as $Z = \lceil log_2(L) \rceil$. If we randomly chose a zoom level from range $[0, Z]$ as $z$, we computed the length of the query condition range as $l = max(L/2^z, 1)$. Similarly, for the `coordinates` attribute, we used the exact coordinates in the sampled tweet as the center. We randomly chose a zoom level and generated a spatial bounding box as the spatial range condition for the query.

In the experiments, we divided the queries into three disjoint sets: a training set, a validation set and an evaluation set. We used a hold-out validation strategy to choose the best agent. When evaluating different approaches, the "difficulty" of the queries in the evaluation workload played an important role. That is, if none of the physical plans of a query were viable, then no approach can generate a viable plan without approximation. On the contrary, if a high percentage (e.g., over 50%) of the physical plans are viable, it would be easy for any method — even a trivial one that picks plans at random — to find a viable plan. In this regard, we further divided the evaluation workload into subsets of queries based on their difficulty measured by the number of viable plans. In our evaluation, we focus on "difficult" queries where less than 50% plans are viable since they can better distinguish the performance of different methods. More evaluation data can be found in the full version [3] of the paper.

**QTE implementations**. We implemented two QTEs to evaluate the Maliva's performance. 1) *Accurate-QTE*. To isolate the effect of estimation errors on the Maliva' performance, we used the actual execution time of the hinted queries as the estimation, and set up a unit cost parameter to represent the time of collecting the selectivity value of one filtering condition in a given rewritten query. Unless otherwise stated, we used 40$ms$ as the unit cost of collecting one selectivity value for the *Accurate-QTE*. 2) We also implemented the ML-based *approximate-QTE* as presented in Section 4.2. We used a random sample table [67] to estimate the selectivity values of query conditions. The selectivity values were used by the approximate-QTE's ML model to estimate the execution time of queries.

**Performance metrics.** We used two metrics to evaluate the performance of different approaches. Recall that a generated rewritten query is "*viable*" if its total response time (including both the planning time and the querying time) is within a given time budget. The "viable query percentage" (VQP) of a solution was the ratio of viable queries over all the queries in the workload. The other metric was called "*Average Query Response Time*" (AQRT), which was the average total response time of all the queries in the workload.

**Query-rewriting Approaches.** We compared the proposed MDP-based approaches with three related methods, i.e., baseline, naive, and Bao [34]. MDP-based approaches included an MDP agent using an approximate-QTE, i.e., MDP (Approximate-QTE), and an MDP agent using an accurate QTE, i.e., MDP (Accurate-QTE). In the baseline approach, the middleware relies on the database optimizer to generate a physical plan for the original query. In the naive approach, we used the same approximate QTE as the MDP-based approach, but enumerated all possible RQs in a brute-force way, then chose the best RQ as the output. The third approach was Bao [34]. We used its open-source release [4] as the server, which provided interfaces for training the model and using the model to choose the best plan for a given set of query plans. Its original client, which was a PostgreSQL plug-in, did not support query hints for using a specific index, which were required by our visualization queries. To solve this problem, we implemented a new client in Python to support such query hints while keeping their server implementation.

In the experiments, we ran both the database and the middleware on the same AWS t2.xlarge instance with four vCPUs, 16GB RAM, and a 500GB SSD drive. We implemented the middleware in Python 3.6 and the neural network using Pytorch 1.7. We evaluated Maliva on both PostgreSQL and a commercial database. All figures were results on PostgreSQL if not stated otherwise.

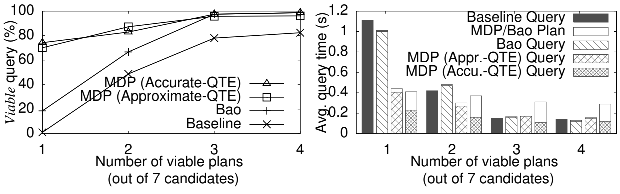
## 7.2 Performance on Using Query Hints

We evaluated the performance of Maliva for only considering query hints in rewriting options (i.e., no approximations). For each dataset, we generated queries with three filtering conditions and set up the rewrite-option set with 8 query-hint sets, i.e., using or not using the index on each attribute. Since one of the 8 hint sets was "no hint at all", which was the original query, the total number of candidate physical plans was 7, i.e., the original query's physical plan was one of the 7 hinted queries. We varied the evaluation workloads with different numbers of viable plans (i.e., 1 – 4 out of 7), and collected the VQP and AQRT metrics for each approach. Table 2 shows the number of queries in the evaluation workloads.

Figures 11(a), 12(a), and 12(a) show viable-query percentages (VQP) on the three datasets. The MDP-based approaches and Bao outperformed the baseline approach significantly, with MDP

**Table 2:** Number of queries in evaluation workloads.

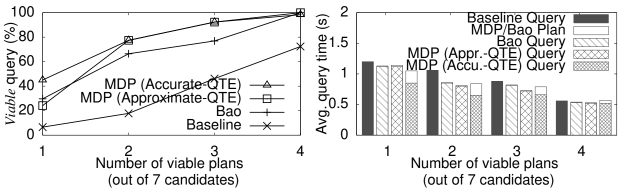
| # of viable plans | 0 | 1 | 2 | 3 | 4 | ≥ 5 |
|---|---|---|---|---|---|---|
| **Twitter** | 518 | 97 | 234 | 118 | 153 | 69 |
| **NYC Taxi** | 408 | 91 | 146 | 13 | 181 | 3 |
| **TPC-H** | 381 | 107 | 310 | 66 | 47 | 0 |




**(a)** Viable query percentage. **(b)** Avg. query response time.

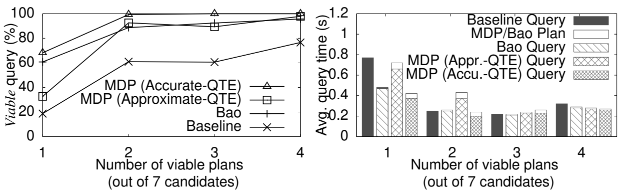
**Figure 11:** Performance on the Twitter dataset ($\tau = 500ms$).

(Accurate-QTE) as the best. For example, on the Twitter dataset, for the queries with a single viable plan, both MDP-based approaches increased the VQP from the baseline's 1% and Bao's 20% to more than 70%. In most cases, MDP (Approximate-QTE) performed better than or comparable to Bao. In one case of the TPC-H dataset, Bao performed better than MDP (Approximate-QTE) mainly because Bao's QTE had a much higher accuracy than the approximate QTE for TPC-H. When the number of viable plans increased from 1 to 4, the VQP of all approaches increased because the more viable plans existed for a query, the easier it was for each approach to find a viable plan in a short amount of time.




**(a)** Viable query percentage. **(b)** Avg. query response time.

**Figure 12:** Performance on the NYC Taxi dataset ($\tau = 1s$).




**(a)** Viable query percentage. **(b)** Avg. query response time.

**Figure 13:** Performance on the TPC-H dataset ($\tau = 500ms$).

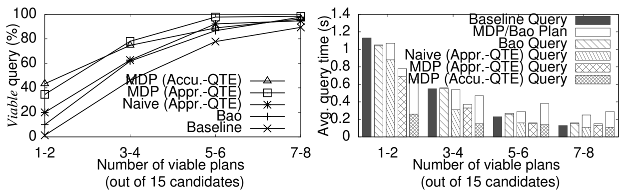
Figures 11(b), 12(b), and 12(b) show the results of the average query-response time (AQRT) of different approaches. On the Twitter dataset, Bao had a comparable AQRT to the baseline, while MDP (Approximate-QTE) had much lower time than the baseline and Bao for queries with one or two viable plans. For example, MDP (Approximate-QTE) reduced the average response time from the baseline's 1.11 seconds and Bao's 1.01 seconds to 0.4 seconds. On the NYC Taxi dataset, Bao and MDP-based approaches had comparable performance and were slightly better than the baseline. On the TPC-H dataset, Bao was better than or comparable to the baseline. In two cases, Bao performed better than MDP (Approximate-QTE) because Bao's QTE had a much higher accuracy than the approximate QTE on TPC-H. However, in all cases, MDP (Accurate-QTE) always had a lower query time than Bao and the baseline, which means it generated a more efficient plan. In cases where MDP (Accurate-QTE) had a longer response time, the extra planning time was the main reason. At the same time, the high VQP of MDP (Accurate-QTE) proved the ability of the MDP model balancing the planning time and the query-execution time to maximize the chance of generating a viable rewritten query.

## 7.3 Effect of Rewrite-Option Number

We evaluated the effect of the number of rewriting options on the Twitter dataset. We set up workloads of queries with different numbers of filtering conditions, resulting in different numbers of rewriting options. To illustrate the planning efficiency of MDP-based approaches, we also evaluated a naive approach, i.e., Naive (Approximate-QTE), which enumerated all possible RQs, estimated their time using the approximate QTE, and chose the best RQ as output. Table 3 shows the number of queries for the workloads. (Due to the space limit, we only show results for 16 rewriting options. More results can be found in the full version [3] of the paper.)

**Table 3:** Workloads with 16 rewriting options.

| # of viable plans | 0 | 1-2 | 3-4 | 5-6 | 7-8 | ≥ 9 |
|---|---|---|---|---|---|---|
| **# of queries** | 485 | 150 | 241 | 90 | 132 | 93 |




**(a)** Viable query percentage. **(b)** Avg. query response time.

**Figure 14:** Performance for 16 ROs on the Twitter dataset ($\tau = 500ms$).

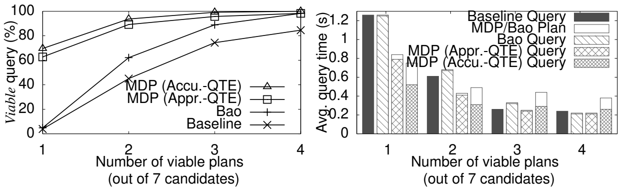
As shown in Figure 14(a), the two MDP approaches performed the best, generating up to 40× more viable queries than both Bao and the baseline approach on queries with one or two viable plans.

Figure 14(b) shows the AQRT results. Consistent with the VQP results, MDP-based approaches outperformed both Bao and the baseline approach. For example, MDP (Approximate-QTE) reduced the average response time from the baseline's 1.13 seconds and Bao's 1.05 seconds to 0.66 seconds for queries with one or two viable plans. Note that in both VQP and AQRT results, the MDP-based approach performed significantly better than the naive approach using the same approximate QTE. These results show the benefit of MDP-based careful planning strategy over a brute-force enumeration approach.

## 7.4 Effect of Time Budget

We evaluated the effect of time budget on the performance of different approaches. We varied the time budget on the Twitter dataset. We show results for 1-second time budget, and more results can be found in the full version [3] of the paper.

As shown in Figure 15(a) and (b), the MDP-based approaches outperformed both Bao and the baseline approach significantly. MDP (Accurate-QTE) outperformed MDP (Approximate-QTE) since the agent could afford the expensive estimation cost for more accurate estimations to find better-rewritten queries. These



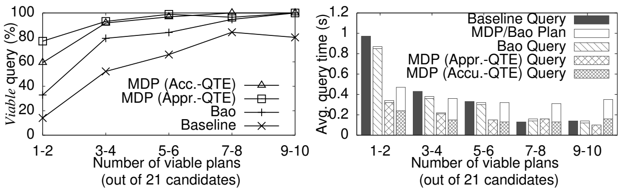

**(a)** Viable query percentage. **(b)** Avg. query response time.

**Figure 15:** Performance for 1-second time budget on the Twitter dataset.

results show that the MDP model is adaptive to QTEs with different costs and accuracies for different time budgets. Compared with the results in Figure 11 where the time budget was 500*ms*, MDP (Accurate-QTE) performed better when the budget was higher, and MDP (Approximate-QTE) performed better when the budget was lower.

## 7.5 Performance on Join Queries

To evaluate the performance of Maliva on queries with joins, we set up a workload of queries joining the `tweets` and `users` tables with filtering conditions on three attributes. For the MDP-based approaches and Bao, we considered 7 different ways of using or not using indexes on the three attributes and 3 different join methods (i.e., nest-loop-join, hash-join, and merge-join) between the two tables. Thus we had 21 query-hint sets in total as the rewriting options. Figure 16(a) shows that for all workloads, the MDP-based approaches outperformed Bao. For the queries with only one or two viable plans, MDP (Approximate-QTE) generated more than twice as many viable plans as Bao. Figure 16(b) shows that MDP (Approximate-QTE) outperformed Bao in all cases. For queries with one or two viable plans, the MDP-based approach reduced the average query response time from Bao's 0.87 second to 0.34 second.




**(a)** Viable query percentage. **(b)** Avg. query response time.

**Figure 16:** Performance for join queries (Twitter dataset, $\tau = 500ms$).

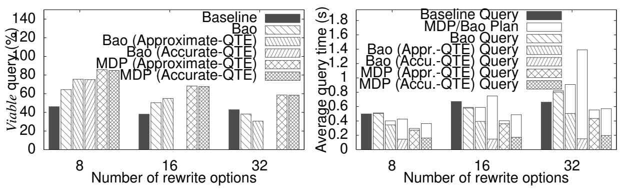
## 7.6 Additional Comparison with Bao

We further compared the performance of Maliva with Bao to demonstrate the advantage of our approach (see Figure 17). Besides the original Bao approach, we included two additional variants — *Bao (Approximate-QTE)* and *Bao (Accurate-QTE)* — that integrated Bao's enumeration strategy on top of our QTEs. We focused on "difficult" queries where less than 50% physical plans were viable. We used the Twitter dataset and varied the number of rewrite options from 8 to 32 (as described in Section 7.3).

**Table 4:** Workloads of queries where less than 50% plans were viable.

| **# of rewrite options** | 8 | 16 | 32 |
|---|---|---|---|
| **# of queries** | 449 | 481 | 497 |

As shown in Figure 17, our MDP-based approaches outperformed both the baseline approach and Bao-based approaches significantly in all the cases. For the 8 rewrite-option workload, both Bao-based approaches using the approximate-QTE and the accurate-QTE outperformed the original Bao. The reason was that Bao's own QTE relied on the plan tree and operators' cost estimations from the physical plan generated by PostgreSQL. As a result, it suffered from the significant estimation errors by PostgreSQL for textual and spatial filtering conditions. With the help of the approximate and accurate QTEs' more accurate estimations, the performance of Bao was improved. However, when the number of rewrite options was 32, both Bao (Approximate-QTE) and Bao (Accurate-QTE) performed even worse than the baseline due to the high cost of estimating all the candidate plans in the brute-force query-planning phase. The VQP of Bao (Accurate-QTE) dropped to 0% because the planning time exceeded the 500*ms* time budget. As shown in the 32 rewrite-option column of Figure 17(b), by judiciously choosing which rewritten queries to run the expensive accurate-QTE, the MDP (Accurate-QTE) reduced the average planning time from Bao (Accurate-QTE)'s 1.24 seconds to 0.37 seconds, with a reduction of more than 70%. This result showed the superiority of using the MDP-based approach for query planning over Bao's brute-force approach.

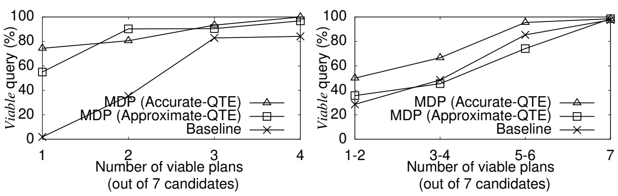



**(a)** Viable query percentage. **(b)** Avg. query response time.

**Figure 17:** Comparison with Bao on the Twitter dataset ($\tau = 500ms$).

## 7.7 Unseen Queries and Other Databases

To evaluate how well Maliva can be generalized to handle unseen queries, we did experiments on the Twitter dataset to train and test the MDP model using two workloads with different query shapes. The training queries were on a single `tweets` table with three filtering conditions. In comparison, the testing queries joined the `tweets` table and the `users` table on `user_id` with three filtering conditions on the former table. As shown in Figure 18(a), the MDP-based approaches outperformed the baseline significantly on the workload with unseen queries. For example, for queries with a single viable plan, the MDP (Approximate-QTE) approach increased the VQP from the baseline's 2% to 55%, and the MDP (Accurate-QTE) approach further increased it to 74%.



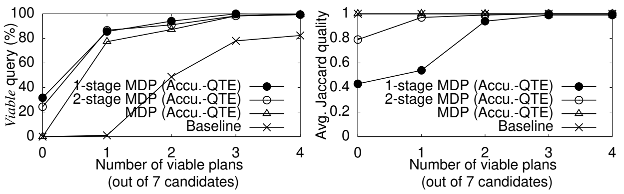

**(a)** Unseen queries ($\tau = 500ms$). **(b)** Commercial DB ($\tau = 250ms$).

**Figure 18:** Generalization to (a) handle unseen queries and (b) use a commercial database.

We also did experiments on the Twitter dataset using a commercial database. We used a smaller table with 10 million records and thus a smaller time budget (250*ms*). The result is shown in Figure18(b). Due to the commercial database's complex behaviors, the approximate QTE had a much lower accuracy (two orders of magnitude) than it had on PostgreSQL. The reason was the approximate QTE only considered predicates' selectivities for estimation, but more factors in the commercial database affected the query time, such as buffering and dynamic execution plan change. However, MDP (Approximate QTE) still had comparable performance (VQP) to the baseline. With a more accurate yet more expensive QTE, MDP (Accurate-QTE) outperformed the baseline for all the queries. For example, for queries with one or two viable plans, the baseline had a VQP of 23%, MDP (approximate-QTE) had a VQP of 36%, and MDP (Accurate-QTE) increased the VQP to 50%.

## 7.8 Performance of Quality-Aware Rewriting

We evaluated the performance of the two quality-aware query rewriting approaches (i.e., one-stage and two-stage) described in Section 6. We used the same Twitter dataset and workload as in Section 7.2. We compared them with the baseline approach and the MDP approach without considering approximation rules. For the quality-aware rewriting approaches, we considered five approximation rules (i.e., adding a `LIMIT` clause with 0.032%, 0.16%, 0.8%, 4%, and 20% of the estimated cardinality of the query) in addition to the eight query-hint sets considered in Section 7.2. All MDP approaches used an accurate-QTE. Besides the AQP and AQRT metrics, we collected a new metric called *Jaccard-based Quality*, which computed the Jaccard similarity between the visualization result of a rewritten query and that of the original query.



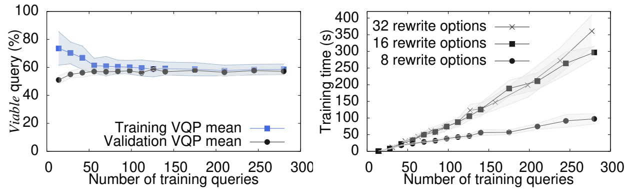

**(a)** Viable query percentage. **(b)** Avg. Jaccard-based quality.

**Figure 19:** Performance of quality-aware rewriting (Twitter, $\tau = 500ms$).

Figure 19(a) shows the VQP of these approaches. For the group of queries without any viable plan, the MDP approach without considering approximation rules and the baseline approach had a zero VQP. By generating approximate rewritten queries, the two-stage MDP approach increased the VQP to 24%, and the one-stage MDP approach further increased the VQP to 31%. There were 518 queries in the 0-viable-plan workload (Table 2), and the one-stage MDP approach generated more than 35 viable queries than the two-stage approach. In terms of efficiency, the one-stage MDP approach outperformed the two-stage approach in all cases. Figure 19(b) shows the average Jaccard-based quality of the rewritten queries generated by different approaches. Both the baseline and the MDP approach without considering approximation rules had no quality loss. The two-stage MDP approach had a significant advantage over the one-stage approach in terms of quality. For example, The former increased the quality of the 0-viable-plan queries from the one-stage approach's 0.43 to 0.79.

## 7.9 Training Performance

We evaluated the training performance for workloads with different numbers of rewriting options on the Twitter dataset. For each workload, we divided a set of about 1,400 queries into a training set and a validation set. Then we varied the number of training queries and randomly sampled those from the training set without replacement. We then used the sampled queries to train an MDP agent and tested its performance on both the training queries and the validation queries. We repeated the step ten times for each number of training queries and collected the mean and standard deviation of the VQPs. We conducted the experiments on the MDP approach using the Accurate-QTE. We show results for 8 rewriting options, and more results can be found in the full version [3] of the paper.




**(a)** Learning curve for 8 ROs. **(b)** Training time for different # of ROs.

**Figure 20:** Learning curve and training time on the Twitter dataset. The shaded area is plotted with "mean + standard deviation" as the upper bound and "mean − standard deviation" as the lower bound.

Figure 20(a) shows the trend when we varied the number of training queries. The VQP on the validation set was close to the VQP on the training set for about 50 training queries. Figure 20(b) shows the training time of different numbers of rewrite options on the training sizes. For the same number of training queries, more rewrite options resulted in a larger q-network, which took more time to update the weights. For the workload with thirty-two rewrite options, it took about 150 seconds to train an MDP agent on 150 training queries.

**Remarks:** The experiments show that Maliva outperformed the baseline in terms of both the number of viable queries and average query response time. Maliva generated up to 70× more viable queries than the baseline. The advantages of Maliva were shown in both the real and synthetic datasets, for different numbers of rewriting options, time budgets and query workloads. Its offline training overhead was relatively small. By considering approximation rules, Maliva generated even more viable queries. The comparison with Bao shows the advantage of Maliva due to the fact these two techniques were designed with different settings and optimization goals.

**Limitations:** One limitation of Maliva is that when the number of rewriting options was significant (e.g., ≥ 32), both the training and the online planning overhead of the MDP models became expensive. Also, for different sets of rewriting options, Maliva requires training different models.

## 8 CONCLUSIONS

In this paper we studied how to rewrite database queries to improve execution performance in middleware-based visualization systems. We explored two optimization options of adding hints and doing approximation. We developed a novel solution called Maliva, which adopts a Markov Decision Process (MDP) model to rewrite a visualization request under a tight time constraint. We gave a full specification of the solution, including how to construct an MDP model, how to train an agent, and how to use approximating rewriting options. Our experiments on both real and synthetic datasets showed that Maliva performed significantly better than the baseline without no-rewriting options in terms of both the probability of serving a visualization request within a time budget and query execution time.

## ACKNOWLEDGMENTS

This work was supported by a UCI ICS research award and an award from the Orange County Health Care Agency.

# REFERENCES


[1] Sye-Min Chan ands Ling Xiao, John Gerth, and Pat Hanrahan. 2008. Maintaining interactivity while exploring massive time series. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology, IEEE VAST 2008, Columbus, Ohio, USA, 19-24 October 2008*. https://doi.org/10.1109/VAST.2008.4677357

[2] AsterixDB Query Hints. http://asterixdb.apache.org/docs/0.9.6/sqlpp/manual.html#Query_hints.

[3] Maliva: Using Machine Learning to Rewrite Visualization Queries Under Time Constraints. arXiv:cs.DB/2112.00182

[4] BaoForPostgreSQL. https://github.com/learnedsystems/baoforpostgresql.

[5] Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic Prefetching of Data Tiles for Interactive Visualization. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). https://doi.org/10.1145/2882903.2882919

[6] Leilani Battle, R. Jordan Crouser, Audace Nakeshimana, Ananda Montoly, Remco Chang, and Michael Stonebraker. 2020. The Role of Latency and Task Complexity in Predicting Visual Search Behavior. *IEEE Trans. Vis. Comput. Graph.* 26, 1 (2020). https://doi.org/10.1109/TVCG.2019.2934556

[7] Mihai Budiu, Parikshit Gopalan, Lalith Suresh, Udi Wieder, Han Kruiger, and Marcos K. Aguilera. 2019. Hillview: A trillion-cell spreadsheet for big data. *PVLDB* 12, 11 (2019). http://www.vldb.org/pvldb/vol12/p1442-budiu.pdf

[8] Daniel Cheng, Peter Schretlen, Nathan Kronenfeld, Neil Bozowsky, and William Wright. 2013. Tile based visual analytics for Twitter big data exploratory analysis. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*. https://doi.org/10.1109/BigData.2013.6691787

[9] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2015. Vizdom: Interactive Analytics through Pen and Touch. *PVLDB* 8, 12 (2015). https://doi.org/10.14778/2824032.2824127

[10] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2016. The case for interactive data exploration accelerators (IDEAs). In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Carsten Binnig, Alan Fekete, and Arnab Nandi (Eds.). https://doi.org/10.1145/2939502.2939513

[11] Cicero Augusto de Lara Pahins, Sean A. Stephens, Carlos Scheidegger, and João Luiz Dihl Comba. 2017. Hashedcubes: Simple, Low Memory, Real-Time Visual Exploration of Big Data. *IEEE Trans. Vis. Comput. Graph.* 23, 1 (2017). https://doi.org/10.1109/TVCG.2016.2598624

[12] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. 2016. Sample + Seek: Approximating Aggregates with Distribution Precision Guarantee. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. https://doi.org/10.1145/2882903.2915249

[13] Ahmed Eldawy, Mohamed F. Mokbel, and Christopher Jonathan. 2016. HadoopViz: A MapReduce framework for extensible visualization of big spatial data. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. https://doi.org/10.1109/ICDE.2016.7498274

[14] Danyel Fisher, Igor O. Popov, Steven M. Drucker, and m. c. schraefel. 2012. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*. https://doi.org/10.1145/2207676.2208294

[15] Parke Godfrey, Jarek Gryz, and Piotr Lasek. 2016. Interactive Visualization of Large Data Sets. *IEEE Trans. Knowl. Data Eng.* 28, 8 (2016). https://doi.org/10.1109/TKDE.2016.2557324

[16] Tao Guo, Kaiyu Feng, Gao Cong, and Zhifeng Bao. 2018. Efficient Selection of Geospatial Data on Maps for Interactive and Visualized Exploration. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. https://doi.org/10.1145/3183713.3183738

[17] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). https://doi.org/10.1145/3318464.3389741

[18] Hint(SQL. https://en.wikipedia.org/wiki/Hint_(SQL).

[19] Kevin Zeng Hu, Michiel A. Bakker, Stephen Li, Tim Kraska, and César A. Hidalgo. 2019. VizML: A Machine Learning Approach to Visualization Recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). https://doi.org/10.1145/3290605.3300358

[20] Jean-Francois Im, Felix Giguere Villegas, and Michael J. McGuffin. 2013. VisReduce: Fast and responsive incremental information visualization of large datasets. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, Xiaohua Hu, Tsau Young Lin, Vijay V. Raghavan, Benjamin W. Wah, Ricardo A. Baeza-Yates, Geoffrey C. Fox, Cyrus Shahabi, Matthew Smith, Qiang Yang, Rayid Ghani, Wei Fan, Ronny Lempel, and Raghunath Nambiar (Eds.). https://doi.org/10.1109/BigData.2013.6691710

[21] Jia Yu and Mohamed Sarwat. 2020. Accelerating Spatial Data Visualization Dashboards via a Materialized Sampling Approach. In *Proceedings of the International Conference on Data Engineering, ICDE*.

[22] Lilong Jiang, Protiva Rahman, and Arnab Nandi. 2018. Evaluating Interactive Data Systems: Workloads, Metrics, and Guidelines. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. https://doi.org/10.1145/3183713.3197386

[23] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. 2014. Distributed and interactive cube exploration. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, Isabel F. Cruz, Elena Ferrari, Yufei Tao, Elisa Bertino, and Goce Trajcevski (Eds.). https://doi.org/10.1109/ICDE.2014.6816674

[24] Tim Kraska. 2018. Northstar: An Interactive Data Science System. *PVLDB* 11, 12 (2018). http://www.vldb.org/pvldb/vol11/p2150-kraska.pdf

[25] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *CoRR* abs/1808.03196 (2018). arXiv:1808.03196 http://arxiv.org/abs/1808.03196

[26] Doris Jung Lin Lee and Aditya G. Parameswaran. 2018. The Case for a Visual Discovery Assistant: A Holistic Solution for Accelerating Visual Data Exploration. *IEEE Data Eng. Bull.* 41, 3 (2018). http://sites.computer.org/debull/A18sept/p3.pdf

[27] Kaiyu Li and Guoliang Li. 2018. Approximate Query Processing: What is New and Where to Go? - A Survey on Approximate Query Processing. *Data Science and Engineering* 3, 4 (2018). https://doi.org/10.1007/s41019-018-0074-4

[28] Lauro Didier Lins, James T. Klosowski, and Carlos Eduardo Scheidegger. 2013. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Trans. Vis. Comput. Graph.* 19, 12 (2013). https://doi.org/10.1109/TVCG.2013.179

[29] Zhicheng Liu and Jeffrey Heer. 2014. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Trans. Vis. Comput. Graph.* 20, 12 (2014). https://doi.org/10.1109/TVCG.2014.2346452

[30] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. 2013. *imMens*: Real-time Visual Querying of Big Data. *Comput. Graph. Forum* 32, 3 (2013). https://doi.org/10.1111/cgf.12129

[31] G. Lohman. 2014. Is Query Optimization a "Solved" Problem? *ACM SIGMOD Blog*. ACM Blog, 14' (2014).

[32] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. 2020. Interactive Cleaning for Progressive Visualization through Composite Questions. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. https://doi.org/10.1109/ICDE48307.2020.00069

[33] Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018. DeepEye: Creating Good Data Visualizations by Keyword Search. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). https://doi.org/10.1145/3183713.3193545

[34] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2020. Bao: Learning to Steer Query Optimizers. *CoRR* abs/2004.03814 (2020). arXiv:2004.03814 https://arxiv.org/abs/2004.03814

[35] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). https://doi.org/10.1145/3448016.3452838

[36] 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, aiDM@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, Rajesh Bordawekar and Oded Shmueli (Eds.). https://doi.org/10.1145/3211954.3211957

[37] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (2019). https://doi.org/10.14778/3342263.3342644

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

[39] Dominik Moritz, Danyel Fisher, Bolin Ding, and Chi Wang. 2017. Trust, but Verify: Optimistic Visualizations of Approximate Queries for Exploring Big Data. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017*. https://doi.org/10.1145/3025453.3025456

[40] Dominik Moritz, Bill Howe, and Jeffrey Heer. 2019. Falcon: Balancing Interactive Latency and Resolution Sensitivity for Scalable Linked Visualizations. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*. https://doi.org/10.1145/3290605.3300924

[41] Barzan Mozafari, Jags Ramnarayan, Sudhir Menon, Yogesh Mahajan, Soubhik Chakraborty, Hemant Bhanawat, and Kishor Bachhav. 2017. SnappyData: A Unified Cluster for Streaming, Transactions and Interactice Analytics. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. http://cidrdb.org/cidr2017/papers/p28-mozafari-cidr17.pdf

[42] MySQL Optimizer Hints. https://dev.mysql.com/doc/refman/8.0/en/optimizer-hints.html.
[43] NYC Taxi Data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.
[44] Oracle Using Optimizer Hints. https://docs.oracle.com/cd/B19306_01/server.102/b14211/hintsref.htm#i8327.
[45] Yongjoo Park, Michael J. Cafarella, and Barzan Mozafari. 2016. Visualization-aware sampling for very large databases. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016.* https://doi.org/10.1109/ICDE.2016.7498287
[46] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. VerdictDB: Universalizing Approximate Query Processing. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.* https://doi.org/10.1145/3183713.3196905
[47] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. 2020. QuickSel: Quick Selectivity Learning with Mixture Models. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). https://doi.org/10.1145/3318464.3389727
[48] Jinglin Peng, Dongxiang Zhang, Jiannan Wang, and Jian Pei. 2018. AQP++: Connecting Approximate Query Processing With Aggregate Precomputation for Interactive Analytics. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.* https://doi.org/10.1145/3183713.3183747
[49] PostgreSQL Query Hints. https://pghintplan.osdn.jp/pg_hint_plan.html.
[50] Fotis Psallidas and Eugene Wu. 2018. Provenance for Interactive Visualizations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2018, Houston, TX, USA, June 10, 2018.* https://doi.org/10.1145/3209900.3209904
[51] Xin Qian, Ryan A. Rossi, Fan Du, Sungchul Kim, Eunyee Koh, Sana Malik, Tak Yeon Lee, and Joel Chan. 2020. ML-based Visualization Recommendation: Learning to Recommend Visualizations from Data. *CoRR* abs/2009.12316 (2020). arXiv:2009.12316 https://arxiv.org/abs/2009.12316
[52] Sajjadur Rahman, Maryam Aliakbarpour, Hidy Kong, Eric Blais, Karrie Karahalios, Aditya G. Parameswaran, and Ronitt Rubinfeld. 2017. I've Seen "Enough": Incrementally Improving Visualizations to Support Rapid Decision Making. *PVLDB* 10, 11 (2017). https://doi.org/10.14778/3137628.3137637
[53] Elke A. Rundensteiner, Matthew O. Ward, Zaixian Xie, Qingguang Cui, Charudatta V. Wad, Di Yang, and Shiping Huang. 2007. Xmdvtool$^Q$: : quality-aware interactive data exploration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007.* https://doi.org/10.1145/1247480.1247623
[54] Sourav Sikdar and Chris Jermaine. 2020. MONSOON: Multi-Step Optimization and Execution of Queries with Partially Obscured Predicates. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). https://doi.org/10.1145/3318464.3389728
[55] SQL Server Query Hints. https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver15.
[56] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *Proc. VLDB Endow.* 13, 3 (2019). https://doi.org/10.14778/3368289.3368296
[57] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning - an introduction.* MIT Press. https://www.worldcat.org/oclc/37293240
[58] Wenbo Tao, Xiaoyu Liu, Yedi Wang, Leilani Battle, Çagatay Demiralp, Remco Chang, and Michael Stonebraker. 2019. Kyrix: Interactive Pan/Zoom Visualizations at Scale. *Comput. Graph. Forum* 38, 3 (2019). https://doi.org/10.1111/cgf.13708
[59] Michel Tokic. 2010. Adaptive epsilon-Greedy Exploration in Reinforcement Learning Based on Value Difference. In *KI 2010: Advances in Artificial Intelligence, 33rd Annual German Conference on AI, Karlsruhe, Germany, September 21-24, 2010. Proceedings (Lecture Notes in Computer Science)*, Rüdiger Dillmann, Jürgen Beyerer, Uwe D. Hanebeck, and Tanja Schultz (Eds.), Vol. 6359. https://doi.org/10.1007/978-3-642-16111-7_23
[60] tpch. TPC-H Website, http://www.tpc.org/tpch/.
[61] Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. 2019. SkinnerDB: Regret-Bounded Query Evaluation via Reinforcement Learning. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). https://doi.org/10.1145/3299869.3300088
[62] Lu Wang, Robert Christensen, Feifei Li, and Ke Yi. 2015. Spatial Online Sampling and Aggregation. *PVLDB* 9, 3 (2015). https://doi.org/10.14778/2850583.2850584
[63] Qianwen Wang, Zhutian Chen, Yong Wang, and Huamin Qu. 2020. Applying Machine Learning Advances to Data Visualization: A Survey on ML4VIS. *CoRR* abs/2012.00467 (2020). arXiv:2012.00467 https://arxiv.org/abs/2012.00467
[64] Yunhai Wang, Kang Feng, Xiaowei Chu, Jian Zhang, Chi-Wing Fu, Michael Sedlmair, Xiaohui Yu, and Baoquan Chen. 2018. A Perception-Driven Approach to Supervised Dimensionality Reduction for Visualization. *IEEE Trans. Vis. Comput. Graph.* 24, 5 (2018). https://doi.org/10.1109/TVCG.2017.2701829
[65] Zhe Wang, Nivan Ferreira, Youhao Wei, Aarthy Sankari Bhaskar, and Carlos Scheidegger. 2017. Gaussian Cubes: Real-Time Modeling for Visual Exploration of Large Multidimensional Datasets. *IEEE Trans. Vis. Comput. Graph.* 23, 1 (2017). https://doi.org/10.1109/TVCG.2016.2598694
[66] Christopher J. C. H. Watkins and Peter Dayan. 1992. Technical Note Q-Learning. *Mach. Learn.* 8 (1992). https://doi.org/10.1007/BF00992698
[67] Wentao Wu, Yun Chi, Shenghuo Zhu, Jun'ichi Tatemura, Hakan Hacigümüs, and Jeffrey F. Naughton. 2013. Predicting query execution time: Are optimizer cost models really unusable?. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou (Eds.). https://doi.org/10.1109/ICDE.2013.6544899
[68] Wentao Wu, Xi Wu, Hakan Hacigümüs, and Jeffrey F. Naughton. 2014. Uncertainty Aware Query Execution Time Prediction. *PVLDB* 7, 14 (2014).
[69] Jia Yu, Raha Moraffah, and Mohamed Sarwat. 2017. Hippo in Action: Scalable Indexing of a Billion New York City Taxi Trips and Beyond. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017.* https://doi.org/10.1109/ICDE.2017.201
[70] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. 2018. GeoSparkViz: a scalable geospatial data visualization framework in the apache spark ecosystem. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management, SSDBM 2018, Bozen-Bolzano, Italy, July 09-11, 2018.* https://doi.org/10.1145/3221269.3223040
[71] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement Learning with Tree-LSTM for Join Order Selection. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020.* https://doi.org/10.1109/ICDE48307.2020.00116
[72] Kai Zeng, Sameer Agarwal, Ankur Dave, Michael Armbrust, and Ion Stoica. 2015. G-OLA: Generalized On-Line Aggregation for Interactive Analysis on Big Data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* https://doi.org/10.1145/2723372.2735381
[73] Xuhong Zhang, Jun Wang, Jiangling Yin, and Shouling Ji. 2016. Sapprox: Enabling Efficient and Accurate Approximations on Sub-datasets with Distribution-aware Online Sampling. *PVLDB* 10, 3 (2016). https://doi.org/10.14778/3021924.3021928