Holistic query Approximation via RL Modeling

Susan B. Davidson
University of Pennsylvania
susan@cis.upenn.edu

Tova Milo
Tel Aviv University
milo@post.tau.ac.il

Kathy Razmadze
Tel Aviv University
kathyr@mail.tau.ac.il

Gal Zeevi
Tel Aviv University
galzeevi@mail.tau.ac.il

ABSTRACT

In data exploration, executing queries over a large database can be time-consuming. Previous work has proposed approximate query processing as a way to speed up aggregate queries in this context, but do not address non-aggregate queries. Our paper introduces a novel holistic approach to handle both types of queries by finding an optimized subset of data, referred to as an *approximation set*. The goal is to maximize query result quality while using a smaller set of data, thereby significantly reducing the query execution time. We formalize this problem as Holistic Approximate Query Processing and establish its NP-completeness. To tackle this, we propose an approximate solution using *Reinforcement Learning*, termed HARLM. While HARLM does not provide theoretical guarantees due to its reliance on Reinforcement Learning, it effectively overcomes challenges related to the large action space and the need for generalization beyond a known query workload. Experimental results on both non-aggregate and aggregate benchmarks show that HARLM significantly outperforms the baselines both in terms of accuracy (30% improvement) and efficiency (10-35X).

PVLDB Reference Format:
Susan B. Davidson, Tova Milo, Kathy Razmadze, and Gal Zeevi. Holistic query Approximation via RL Modeling. PVLDB, 18(6): 1635 - 1648, 2025.
doi:10.14778/3725688.3725695

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/GalZeevi/HARLM.

1 INTRODUCTION

During the data exploration phase of data science, users frequently query iteratively to better understand the data [5]. However, when the database is large and queries are complex, obtaining precise answers for exploratory queries can be time-consuming, leading users to favor faster, yet approximate, query results. While much work has been done in Approximate Query Processing (AQP) [28], existing approaches focus on aggregate queries and overlook complex non-aggregate select-project-join (SPJ) queries. However, such queries may constitute a significant portion of data exploration tasks, leading to unacceptable delays during the exploration phase.

Example 1.1. In a study of IMDB exploratory data analysis (EDA) sessions [26], approximately 50% of the queries were complex SPJ (Select-Project-Join) queries. One such query explores the relationship between production companies' countries of origin and the cast size of movies, requiring joins across five tables with multiple filters. These types of queries are crucial for analysts to understand relationships within the data and are often iteratively refined. This scenario extends beyond just the entertainment industry. For instance, in the healthcare sector, analysts query patient records to identify correlations between treatments and recovery times, joining multiple tables with sensitive and extensive data [27]. Similarly, in the financial industry, analysts run complex queries to detect fraud or assess risks, joining transaction records, customer profiles, and external data sources [28].

Despite efforts to limit result sizes, processing these non-aggregate queries can be time-consuming. In the IMDB study, non-aggregate queries had an average execution time of 25 minutes, with 30% surpassing the one-hour mark. Such lengthy queries are not uncommon, as they often lie outside the optimized query workload that the database was designed to handle. Therefore, as with AQP techniques for aggregate queries, prioritizing fast, approximate responses for *non-aggregate queries* is crucial to enable fast exploratory analysis [3] □

Note that other efforts, such as OLAP, have also focused on accelerating query execution to support faster decision-making [10], and various database architectures have been introduced to facilitate faster data extraction upon demand [40]. However, these approaches are unsuitable for our purposes since they have large storage requirements, require specific expertise to use, and require lengthy setup (see Section 2). We therefore propose a *holistic* approach to AQP which accelerates the processing of complex, aggregate *and non-aggregate* queries while maintaining high accuracy. Given a database and expected query workload, our approach finds a subset of the data in each of the database tables, called an *approximation set*, over which queries are executed to provide fast, but approximate, answers. To do this, we must address two problems: (1) defining what a reasonable approximation is, and (2) efficiently computing the approximation set.

Approximate answers to non-aggregate queries. While approximate answers are well understood for aggregate queries, extending this idea to non-aggregate queries requires careful consideration. Our premise is that an approximation for a non-aggregate query should include some, *but not necessarily all*, tuples in the query result. The rationale for this is that when the query output becomes excessively large, users cannot look closely at the entire result beyond a certain *frame size*, and may not therefore discern if it is approximated. Unlike AQP for aggregate queries, where the quality metric for approximation remains constant across queries, the critical factor in non-aggregate queries is the size of the result that a user can effectively analyze. In scenarios where the user can analyze the entire result, the approximation must be highly accurate, covering the entire result set. However, for queries with

Proceedings of the VLDB Endowment, Vol. 18, No. 6 ISSN 2150-8097.
doi:10.14778/3725688.3725695

exceptionally large results, the approximation can provide the user with a query answer limited to the frame size (see Section 2).

Efficiently computing approximation sets. To efficiently calculate a good approximation set for a query workload, a number of challenges must be addressed: (C1) The vast combinatorial solution space, created by combinations of tuples from different tables. (C2) Understanding the search space, due to the cost of executing the workload. (C3) The varying size of query results, which introduces different importance levels for tuples (e.g. tuples from queries with smaller result sizes are more significant than ones from larger result sizes). (C4) Generalizing for future, potentially dissimilar queries from the known workload. (C5) Detecting when there is a drift in user interest. Although computing the approximation set is time consuming, it can be conducted offline. Analogous to the trade-off identified in AQP, users are willing to accept the compromise between extensive offline pre-processing time and the quick, albeit approximate, results provided online.

Our solution. Our approach creates an approximation set which is then used to provide approximate answers to future queries. To evaluate the quality of an approximation set we establish a quality metric which takes into account the size of the full query result with respect to the frame size for all queries in the workload, weighted by the importance of each query. Since optimizing with respect to this metric is NP-hard, we turn to Reinforcement Learning (RL) to approximate a solution.

Our system, HARLM, starts by revising the query workload to turn aggregate queries into non-aggregate queries by removing aggregate operators. It then pre-processes the database (which may encompass multiple tables) and revised query workload to create a reduced though robust subset of the data (addressing C2, C4). This subset is then translated into the RL action space, where, at each training step, the model predicts the next tuples to include in the approximation set based on the current state. This predictive action aims to maximize the reward, representing the quality of results for queries encountered so far. HARLM employs an RL model utilizing critic-actor networks with Proximal Policy Optimization (PPO), specifically tailored for tabular data. The model encompasses a well-crafted environment and transformed action definitions (addressing C1, C4). A specialized reward function aligns with the goal of selecting rows for accurate approximations (C3).

This yields a robust, adaptive solution that significantly improves the efficiency and accuracy of non-aggregate query approximations. Additionally, metadata for the entire database is retained and combined with the generated approximation set to support aggregation query responses. While HARLM demonstrates significant improvements in accuracy and efficiency, it does not provide mathematical guarantees due to the inherent limitations of Reinforcement Learning methods. However, this aligns with the approach taken by other recent AQP systems (e.g. [42]) that also prioritize empirical effectiveness over formal guarantees. HARLM also detects drift in user interests, enabling model fine-tuning for better approximate results (C5). Our experiments (see Section 6) show that in some cases the computations may take up to an hour. Although this is acceptable in an offline scenario [28], we also give a lightweight version which accomplishes this task much faster (up to 30 minutes) with only a slight degradation in quality. Our user studies substantiate the utility of our system, which ultimately saves users more time than that spent in the computation of approximation sets.

Benefits of our approach. Our solution offers several benefits: (1) An approximation for both aggregate and non-aggregate queries. (2) A versatile mediator adaptable to any database. (3) Flexible training within time constraints, managing the tradeoff between running time and quality. (4) Efficient query interpretation through vector-based techniques, enabling a rapid reduction of the initial space and swift training processes. (5) A predictive model and query interpretation to assess the likelihood of missing answers and enhance result accuracy. (6) The ability to detect interest drift, enabling fine-tuning to evolving exploration needs. These benefits position HARLM as a comprehensive tool for efficient EDA.

Contributions and outline. In addition to providing a comprehensive tool for data exploration, this work is among the first applications of advanced RL concepts. Contributions include:
(1) **Problem Definition and Metric Introduction** (Section 3): We define the problem of approximating non-aggregate queries given a budget for the approximation set, and introduce a novel metric for evaluating its quality.
(2) **Holistic query approximation via RL modeling** (Sections 4): We present a holistic solution based on RL for approximating both aggregate and non-aggregates queries, by creating “good" approximation sets. Additionally, we address challenges such as determining when to use the approximation set for answering queries, when to fine-tune the model in response to a drift in user interests, and how to handle an unknown query workload.
(3) **Experiments Demonstrating Efficiency and Quality** (Section 6): We conduct comprehensive experiments showcasing the efficiency and quality of HARLM for both aggregate and non-aggregate queries. The results highlight its adaptability to diverse time constraints and superior performance in managing the trade-off between running time and result quality.

2 RELATED WORK

Approximate Query Processing (AQP) and Generative Models. AQP typically follows two approaches [28]: (1) retaining a small sample of tuples with metadata, and (2) generating synthetic tuples using generative models. The first includes sampling methods [1, 7, 15], which accelerate query execution by reducing data volume at the cost of precision. While less effective for non-aggregate queries (see Section 6), we draw on [45] to build an initial approximation set based on the query workload for model training. The second approach uses generative models like GANs and VAEs [12, 51] to create synthetic tuples that capture complex distributions. However, such tuples can deviate from real data, leading to empty or misleading query results (see Section 6).

OLAP. Multidimensional OLAP (MOLAP) systems focus on aggregates using data cubes built from past queries [10, 40]. However, they face limitations: (1) high storage costs—up to 10× the original data [27], (2) reliance on OLAP-specific DBA expertise [19], and (3) complex setup procedures that may span days [20]. These factors restrict their usability for typical data scientists.

Data Summarization, Sampling, and Reduction. Representative selection via clustering [29], visualization-aware sampling [44], and query result diversification [55] aim to preserve diversity and relevance. However, as shown in Section 6, these methods are slow and not optimized for query approximation. Sketches and reduction techniques [7, 54] may discard critical details for non-aggregate queries. Other approaches, including skyline [43], caching [2], and view selection [33], do not jointly address aggregate and non-aggregate query needs.

3 PROBLEM DEFINITION AND COMPLEXITY

We now introduce the problem of Holistic Approximate Query Processing (HAQP). We begin by defining the problem in a simplified scenario with a known query workload, and give a metric to assess the quality of a set of subsets of tuples with respect to the given workload. In the next section we extend these concepts to scenarios where the query workload is unknown.

HAQP Problem Definition. Consider a set of relational instances $\mathcal{T} = \{T_1, ..., T_n\}$, an initial query workload Q_I consisting of both aggregate and non-aggregate queries over $\mathcal{T}$, and an importance function $w : Q_I \rightarrow [0, 1]$, where $\sum_{q \in Q_I} w(q) = 1$. By default, the weight is determined by the frequency of query occurrences. Each aggregateine query in Q_I is first transformed into a query without aggregate operations by removing GROUP BY/HAVING clauses as well as the aggregate operations (e.g. SUM, MIN, MAX) from the SELECT clause. Let Q denote the transformed query workload. We assume limits on the size of the available memory (k) as well as on the maximum result size that users can cognitively process (the frame size F). In practice, F may vary from 10 rows (the default data view size in pandas) to 500 rows (the default result size in PostgreSQL), and is configurable.

Metric. Our metric function, $score(S)$, measures the quality of a set of subsets (the *approximation set*) of the relational instances in $\mathcal{T}$, $\mathcal{S} = \{S_1, ..., S_n\}$ where $S_i \subseteq T_i$, with respect to $(\mathcal{T}, Q, w, k, F)$:

$$score(\mathcal{S}) = \sum_{q \in Q} w(q) \cdot min(1, \frac{|q(\mathcal{S})|}{min(F, |q(\mathcal{T})|)}) \quad (1)$$

Intuitively, if a query returns more tuples than the frame size F, there is no need to include in S more than F tuples from the query result since the user may not be able to cognitively process them. Conversely, if a query returns only a few tuples, then each tuple carries significant importance in the result. The metric takes into account the weight assigned to each query and calculates the average score over all queries in Q. Moreover, as this metric is optimized across all queries, the data it retains is sufficiently varied to capture information needed for calculating aggregate queries.

HAQP Goal. Given an instance $\mathcal{T}$, a set of transformed queries Q, weight function w, and positive integers k and F, the goal is to find a set $\mathcal{S}$ where $\sum_{S_i \in \mathcal{S}} |S_i| < k$ such that $score(\mathcal{S})$ is maximized.

Problem Hardness. HAQP is already hard when queries are over a single table, $n = 1$. In this case, a naive approach is to consider all subsets $S \subseteq T$ of size k, and find the one that maximizes $score(S)$, which is computationally intractable. Worse yet, this naive solution cannot be significantly improved since the HAQP problem is NP-hard. This can be shown by reduction from the max-k-vertex cover problem [34]. We create an HAQP instance from the input to the max-k-vertex cover problem as follows: For each vertex $v \in V$, we create a tuple $t_v \in T$, and for each edge $e = \{u, v\} \in E$, we create a query $q_e(T) = \{t_u, t_v\}$ that includes the tuples corresponding to the edge's endpoints $q_e(T) = \{t_u, t_v\}$. The query weights are set to the weights of the corresponding edges in the graph. The limit on the number of informative tuples, F is set to 1, and the memory size is set to k (the input to the max-k-vertex problem). Observe that a solution to the HAQP problem is a solution to the max-k-vertex problem, as the score function $score(S)$ defined in Equation 1 is equivalent to the objective function of the max-k-vertex cover problem. In case of multiple tables ($n > 1$) the problem is even worse combinatorially.

Parameter Settings and unknown queries. As can be seen from the problem formulation, the choice of parameter settings significantly impacts the overall performance of a solution: The frame size (F) directly affects the score function. The available memory (k) determines how large the subsets can be and therefore affects the accuracy and coverage of query results. Increasing k enables a larger number of tuples to be kept, leading to a higher likelihood of obtaining exact query outputs and improved coverage for a wide range of queries. The impact of a variety of different parameters settings are explored in Section 6. Moreover, HAQP was defined for a simplified scenario with a known query workload; however in practice, users may issue new queries which differ from the original workload, or the query workload may be unknown. We address these issues throughout the next section.

Due to the hardness of finding an exact solution to HAQP, as well as the cost of executing queries over large relations to evaluate $score(\mathcal{S})$, we show in the next section how to use Reinforcement Learning (RL) to find a good approximate solution.

4 APPROXIMATION USING RL

In this section we outline our approach, which uses Reinforcement Learning (RL) to solve Holistic Approximate Query Processing (HAQP). After a brief overview of our framework, HARLM, we discuss preprocessing steps for input tables and query workloads, and show how the challenges (C1-C5) presented in the introduction are handled (Section 4.1). A high-level descriptions of the RL training and inference steps are then provided (Sections 4.2 and 4.3). Finally, we discuss additional enhancements, including a "light" version of the system for improved running times and how to handle scenarios without a known query workload (Section 4.4). A deep dive into our use of RL is presented in Section 5.

***Overview of* HARLM** . In our RL approach, an approximation set consisting of tuples from the input tables is learned through trial-and-error interactions. The RL agent starts with some initial state (approximation set), and at each step predicts the next action (tuple selection) based on the current state. This predictive action aims to maximize the reward, representing the quality of results for queries encountered thus far (Equation 1). The use of RL ensures the fitness of selected tuples to the queries, while exploration introduces diversity in tuple selection. The workflow of HARLM is presented in Figure 1. The training phase shown in (a) begins by pre-preprocessing the database D and the query workload Q to

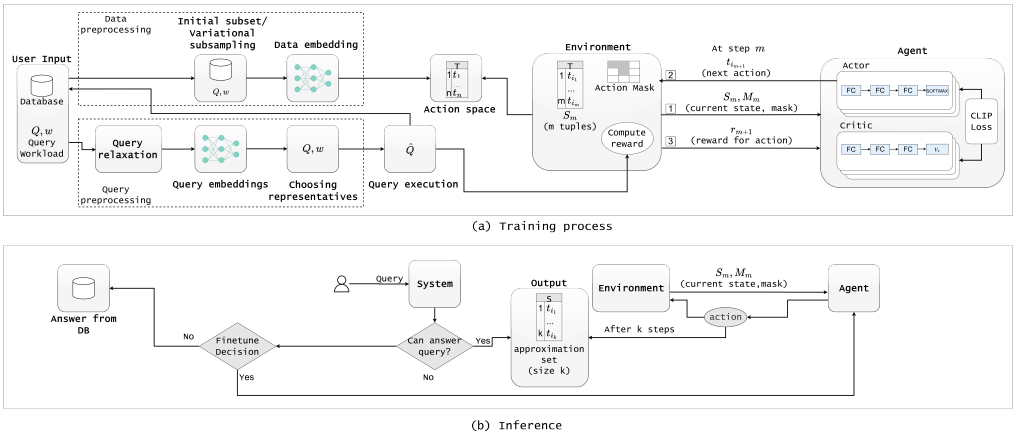

Figure 1: HARLM architecture.

address some of the challenges mentioned in the introduction, and provide input to the RL model. The RL model is then trained. In the inference phase shown in (b), the framework generates an approximation set using the trained model. When a query is issued by the user, the system decides whether to use the approximation set or to query the full database, based on an estimate of how close the query is to those used to train the model.

4.1 Data and Query Pre-processing

The pre-processing phase depicted in Figure 1(a) prepares the data and queries for input to the RL phase and addresses several challenges: the combinatorial size of the solution space ($C1$), and the high cost of executing the query workload ($C2$). We describe how queries and data are pre-processed and how these challenges are addressed in the context of the RL framework. We then discuss how to generalize for future queries ($C4$).

Query Pre-processing. Queries are first generalized to handle potential variations in future workloads before their vector representations are created. After transforming aggregation queries in Q_I into their non-aggregate versions (as discussed in Section 3), *query relaxation* techniques [14] are applied to Q to modify or relax query conditions, particularly when strict conditions lead to sparse results. This relaxation step ensures that the result sets are enlarged, making the queries more general and adaptable to future, unseen queries. To maintain accuracy, we follow the guidelines and best practices outlined in the query relaxation work, ensuring that tuples added during generalization do not degrade the performance of the system. By doing so, we include only relevant tuples beyond those returned by the current workload, enhancing the system's robustness for a broader range of queries.

Vector representations of the generalized queries are then created using a modified version of SENTENCE-BERT [48], specifically tailored for SQL queries. The modification fine-tunes the transformer-based model to understand the syntax and semantics of SQL, allowing it to capture the relationships and structures inherent in queries. By treating SQL queries as text sequences, we embed them into a high-dimensional vector space, positioning semantically similar queries closer together. This process incorporates features such as query operators, conditions, and schema information, enhancing the model's ability to differentiate between various types of queries.

After embedding, we apply a clustering algorithm to group similar queries, identifying what we refer to as *query representatives*. These representatives are then used in the score function, facilitating efficient retrieval and processing of relevant data in the subsequent phases of our approach.

Data Pre-processing. The goal of this phase is to reduce the vast initial action space (i.e., the entire database) and begins after the relaxed query representatives have been selected. The first task is to use the query representatives to query the database, obtaining an initial subset of data by filtering out tuples that do not appear in any query result. Since the query workload has been reduced to just the relaxed query representatives, this is feasible After this, we apply variational subsampling, an AQP technique adapted from [45]. This method subsamples data relevant to the query representatives, further reducing the dataset. For aggregation queries, we also retain metadata from the entire database, in particular histograms of various columns, a practice commonly used in AQP (e.g. [42]). The use of this metadata will be further detailed in Section 4.3. The transformation into vector representations is achieved using a modified version of SENTENCE-BERT[1]. This adaptation ensures that both the structure of the table (column names) and the values are captured in the embedding space, making the vectors suitable for downstream tasks in the RL framework.

Addressing challenges $C1$, $C2$ and $C4$. In the RL framework, challenge $C1$ becomes one of the size of the *action space*. This space constrains the RL algorithm to valid tuple selections aligned with table structures, data distribution, and the provided query workload. Notably, selecting tuples separately from each table may lead to unjoinable tuples [23, 45]. Since a smaller set of tuples is created during the data pre-processing phase based on the generalized query workload, which may contain joins, this problem is avoided. The second challenge, $C2$, is the cost of executing queries in the query

[1]This version is adapted for tabular data by incorporating column names as tokens alongside the row values, allowing the model to capture both the schema and the data content, which enhances the understanding of the relationships between features.

workload, in particular to evaluate the quality of an approximation set. This is addressed during the query pre-processing phase by choosing a smaller set of query representatives, and during the data pre-processing phase by selecting a smaller dataset over which to execute the queries. To overcome the challenge of a future, unknown workload ($C4$), we use a fine-tuning strategy (detailed next). Moreover, the use of query relaxation introduces more distinct tuples into the action space and avoids overfitting to the given query workload. The RL exploration phase also makes it return a more diverse solution, further addressing this challenge.

4.2 Training

Our RL formulation models the problem as a sequential decision-making process. The RL components used during the training phase of HARLM are shown in Figure 1(a) and detailed in Algorithm 1.

Action Space. The database is transformed into an action space which serves as input for the RL model during training. Training involves formulating a policy for selecting actions from this space based on a given state. Since it is infeasible to consider all subsets of tuples across all tables, we create a significantly reduced action space as described in Section 4.1.

State and Environment. The state, a crucial element capturing relevant information for the decision-making policy in Reinforcement Learning (RL) models, is represented as the set of actions previously selected by the model. Recall that an action encompasses multiple tuples sourced from different tables. Thus, the state effectively encapsulates the approximation set chosen thus far. The environment, serving as the external system or context with which the RL agent interacts, defines how states are represented and how the action space is made available to the agent for policy definition. Specifically, we formalize the HARLM environment in *GSL* form, as detailed in Section 5, which builds the chosen subset by allowing the model to add tuples gradually and observing the currently built set at each selection. We also discuss in Section 5 an alternative approach which, due to its limitations, was not incorporated into the algorithm. Nonetheless, the results of the alternative approach are presented in Section 6.

Reward. The reward, furnished by the environment, plays a pivotal role in updating the policy network. Our reward function is the score shown in Equation 1, which is defined by the current batch of queries. To expedite the training process and foster stability in the learned policy, we implement a common technique of batch training and loss computation. Each epoch in the model training uses a distinct batch of queries, directly influencing the reward function. In essence, a high reward signifies that the chosen action (representative of several tuples) encompasses the majority or even all the results from the current batch of queries.

Agent. The agent considers the current state and then selects an action based on the learned policy, π_θ. In HARLM, our agent uses the actor-critic method [36]. This method incorporates several actor networks and critic networks operating asynchronously. The actor networks are used to choose actions based on π_θ, while the critic networks update the policy based on the received reward. The ablation study (Section 6.4) justifies the component of each agent.

Algorithm 1 HARLM Training

Require: Database D, query workload Q
Ensure: A trained RL model
1: $vec\_generalized\_Q = Emb\_sql(relaxation(Q))$
2: $\hat{Q} = rep\_selection(vec\_generalized\_Q)$
3: $\hat{D} = \hat{Q}(D)$
4: $initial\_set = Emb\_tab(variational\_subsampling(\hat{D}))$
5: **for** each $episode$ during training **do**
6: $batch\_Q = batch(\hat{Q})$
7: action a_k is sampled based on $p(a_k|s_k, \theta)$, $a_k \in A$
8: $s_{k+1} = s_k$ union a_k
9: Reward r is then computed based on $state$ and a_k matching to $batch\_Q$
10: Use the reward to calculate loss and update the network
11: **if** early stopping (loss) **then**
12: break
13: **end if**
14: **end for**
15: **return** M

Overall training process. The training pipeline (Figure 1(a)) takes the database D and query workload Q as input and performs a pre-processing step where training queries are generalized and evaluated to construct the action space A. Training proceeds with iterative selection of an approximation set using the RL model. In each epoch, batches of pre-processed queries are used to evaluate actions and update the model.

At step k, the state s_k encodes the effect of k previously selected actions from A. The actor network computes action probabilities $\pi_\theta(a|s) = p(a|s, \theta)$, guiding selection toward actions with higher expected long-term reward. This leads to approximation sets more likely to satisfy the training workload. Multiple actor-critic networks operate asynchronously to explore diverse policies and accelerate training.

Since actions correspond to tuples, we apply action masking [18] to ensure each tuple enters the approximation set at most once. The environment supplies a mask with valid actions at each step. The critic estimates the long-term reward and computes the loss relative to the actual reward, enabling both networks to update for better policy learning (see Section 5).

Addressing Challenges $C3, C4$. In the RL context, challenges $C3$ (diversity and balance in results) and $C4$ (generalization for future queries) necessitate a well-crafted reward system and an effective exploration strategy. $C3$ is addressed by formulating a reward system that carefully balances diversity and relevance. The reward associated with selecting an action for a query with a modest result size holds significant value, emphasizing a balance between diversity and ensuring relevance. This reward mechanism is crucial during the exploitation phase, guiding the selection of actions for the creation of the approximation set. To tackle challenge $C4$ (ensuring selected tuples generalize effectively for future unseen queries), we employ a policy-based RL framework. This framework systematically chooses actions, incorporating an exploration strategy. The exploration strategy aims to explore diverse combinations of actions that are potentially relevant not only to the queries in the current workload but also to future queries. This approach enhances the model's ability to generalize effectively, accommodating a broader spectrum of queries beyond the ones encountered during training.

Algorithm 2 Answerability of queries estimation and Interest drift detection

Require: Approximation Set $\mathcal{S}$, Queries Representatives Emb. $\hat{Q}$, Score $score(\mathcal{S})$ calculated for each $q \in \hat{Q}$, new query nq, global $fineTuningFlag$
Ensure: Action to be performed

for every $\hat{q} \in \hat{Q}$ **do**
  $ED \leftarrow Eucledian\_distance(nq, \hat{q})$
end for
$1nn = \hat{Q}[argmin(ED)]$
$1nn\_dist = min(ED)$
$Estimator\_uncertenty = 1 - [(1 - 1nn\_dist) * score_{1nn}(\mathcal{S})]$
if $Estimator\_uncertenty \leq 0.5$ **then**
  $fineTuningFlag = 0$
  return query the $\mathcal{S}$
end if
if $0.5 \geq Estimator\_uncertenty$ **then**
  if $Estimator\_uncertenty \geq 0.8$ **then**
    $fineTuningFlag \leftarrow fineTuningFlag + 1$
    if $fineTuningFlag = 3$ **then**
      $fineTuningFlag = 0$
      return Issue fine-tuning for the model
    end if
  else
    $fineTuningFlag = 0$
    return Query the DB
  end if
end if

4.3 Inference and User's Interaction

The inference phase occurs after training the RL model (Figure 1(b)). Using the query workload and database, the model outputs the approximation set. Tuple selection for the approximation set is sequential, where groups of tuples are chosen based on the learned policy obtained during model training.

User Interaction and Interest Drift Detection. After generating the initial approximation set, users can issue both aggregate and non-aggregate queries. Algorithm 2 outlines the process for estimating query answerability and detecting interest drift, leveraging the pre-constructed approximation set, query representatives, and their vector representations from the pre-processing phase. Using the scoring function from Section 3, we calculate a score for each query representative within the approximation set.

A global flag tracks potential interest drift, initialized at zero. For each new query, its Euclidean distance to the vector representations of query representatives is computed, identifying the nearest representative and its distance. The system then derives an estimator uncertainty metric, reflecting confidence in using the approximation set for the query. Queries resembling high-scoring representatives can be confidently answered from the approximation set, whereas queries with larger distances may deviate from the training workload, requiring direct database queries to avoid incomplete or biased results. The system responds based on the estimator uncertainty: Low uncertainty (≤ 0.5): Query the approximation set for reliable results; Moderate uncertainty (> 0.5 and < 0.8): Query the database directly for completeness and accuracy; High uncertainty (≥ 0.8): Query the database directly. Additionally, increment the global fine-tuning flag. If three consecutive high-uncertainty queries are observed, the system triggers a fine-tuning process to adapt to emerging workloads.

The fine-tuning process begins with the already trained model, which is re-trained using additional new queries over several episodes. During this phase, the model's policy is adjusted to incorporate the interest drift, resulting in a refined approximation set. Both thresholds—0.5 for querying the database directly and 0.8 for initiating fine-tuning—are experimentally validated and shown to effectively balance accuracy and computational costs (see Section 6). This mechanism addresses challenge $C5$ by detecting and adapting to shifts in user interests. HARLM achieves high accuracy in predicting query answerability (see Section 6.2, Figure 7), reducing risks of incomplete results and user biases.

Handling Aggregation Queries. To handle aggregation queries, we extend the approach outlined in Algorithm 2 by first transforming the aggregation query into its non-aggregate equivalent. If the algorithm decides to query the database or initiate the fine-tuning process, the flow remains consistent with the non-aggregate query handling. However, when the algorithm opts to query the approximation set, we introduce a modification tailored for the aggregation use case.

Intuitively, querying the approximation set mirrors the Open World Query Processing (OWQP) paradigm [41], where only a subset of the data is accessible for querying. In this scenario, data statistics and histograms from the entire dataset can be utilized to estimate query results for the full dataset. We leverage metadata collected during the data pre-processing phase—such as histograms of column combinations—alongside the approximation set to enhance estimation accuracy. Specifically, we assign weights to tuples in the approximation set to adjust results in line with the distribution patterns captured by the metadata. For instance, in sample reweighting, each tuple t in the approximation set is assigned a weight $w(t)$, representing the number of tuples it approximates in the full dataset. Queries are transformed to account for these weights, such as rewriting `COUNT(*)` as `SUM(weight)`.

Since the sample is not uniformly drawn from the data, we employ Iterative Proportional Fitting (IPF), a bias-correction technique, to refine the weights [41, 42]. IPF, inspired by its use in population synthesis within demography [9, 21], calibrates sample weights to align with aggregate population statistics. The procedure iteratively adjusts tuple weights to satisfy the given aggregates. If an aggregate is not satisfied in the sample, the weights of corresponding tuples are rescaled. This iterative process continues until all aggregates are satisfied, converging to an optimal scaling if one exists. In cases where no valid scaling exists, the algorithm provides an approximate reweighting. By integrating this approach, we ensure robust query approximation for aggregation queries, leveraging both the approximation set and pre-processed metadata.

4.4 Further Improvements

HARLM-*light and Adaptive Configuration.* We explored several optimizations to HARLM, resulting in a more efficient version called HARLM-light. These enhancements include an early-stopping threshold, an increased learning rate, and a reduction in the number of queries considered during the pre-processing phase of training. Although this leads to a slight decrease in quality (approximately 10%), the running time is significantly reduced (to just 30 minutes), as shown in Section 6. Despite this decrease in quality, HARLM-light consistently outperforms other competitors. Additionally, by considering user time constraints and the portion of the training query

Baseline	IMDB		MAS	
	Score	setup(m)	Score	setup(m)
HARLM	**0.64±0.06**	60 ± 7	**0.75 ±0.025**	30 ± 2
HARLM-light	0.53±0.09	32 ± 2	0.61 ±0.04	15 ± 0.3
VAE	0.003±0.002	1920 ± 3.5	0.05 ± 0.001	720 ± 3
CACH	0.08±0.06	330 ± 12.5	0.22 ±0.09	34 ± 1.2
RAN	0.29±0.03	0.72 ± 0.08	0.2 ±0.030	0.68 ± 0.02
QUIK	0.34±0.04	160 ± 2	0.25 ±0.03	60 ± 3.1
VERD	0.47±0.02	200 ± 5	0.3 ±0.05	90 ± 2.3
SKY	0.35±0.001	1500 ± 1	0.33 ± 0.001	480 ± 3
BRT	0.3±0.007	2880 ± 0.	0.4 ±0.001	2880 ± 0.01
QRD	0.32±0.06	1800 ± 1	0.38 ±0.03	35 ± 2
TOP	0.27±0.0	338 ± 21	0.46 ±0.0	36 ± 2.3
GRE	0.11 ± 0.0	2880 ± 0.	0.51 ±0.02	2880 ± 0.01

Table 1: Quality and Running time

workload to be used, our system intelligently selects the optimal configuration for each specific use case. This process involves dynamically adjusting various parameters, allowing for flexibility between the most lightweight version and the highest-performing configuration in terms of quality. This adaptive approach optimizes the balance between running time and approximation set quality, ensuring that our system meets user requirements efficiently.

Diversity. While our metric function does not explicitly include diversity, our chosen RL solution incorporates an exploration policy. This feature leads to a diverse solution, enriching the variety of query results provided to the user. In Section 6, we present experiments comparing the diversity of our solution against other methods, showcasing its superiority over competitors.

Handling Unknown Query Workloads. In Section 3, we frame the problem as optimizing a metric function relative to a given workload. To tackle cases where the query workload is unknown, we draw inspiration from prior works that generate synthetic query workloads [24, 32]. HARLM generates query workloads based on statistical information gathered from the tables, such as the mean and standard deviation of numerical columns, a sampled set of categorical columns (with repetition to reflect the popularity of certain values), and predefined query templates. While this method produces a satisfactory initial workload (Section 6), the system also iteratively generates additional queries during interactions with the users that reflect their focus, and refines its model accordingly.

5 REINFORCEMENT LEARNING MODEL IMPLEMENTATION

Our agent's policy is governed by a reinforcement learning model. We begin by introducing the actor-critic network, and show how proximal policy optimization contributes to the creation of a faster and more suitable approximation set (Section 5.1). We then present the selected environment and discuss alternatives along with their limitations (Section 5.2). Notably, this research is the first application of advanced RL concepts, specifically proximal policy optimization, for tabular data purposes. In this domain, the action space is exceptionally large, and the state representation is extremely intricate, posing significant challenges for adapting RL methods.

5.1 Actor-Critic with Proximal Policy Optimization

In HARLM, the agent's policy uses the state representation as input to infer the optimal action. To address high reward variance and improve performance, we adopt an actor-critic method with entropy regularization, which fine-tunes the objective function to promote tuple diversity. Policy-based reinforcement learning methods, such as REINFORCE [22], optimize the parameterized policy π_θ to maximize the expected long-term reward:

$$J(\theta) = E_{\pi_\theta}[R(\tau)] = \sum_{\tau} p(\tau|\theta) \sum_{t=1}^{T} r_t$$

where $\tau = (s_1, a_1, \ldots, s_T, a_T)$ represents a trajectory, and $R(\tau)$ is the cumulative reward for a selected approximation set $approx\_set = [a_1, \ldots, a_T]$. The goal is to increase the probability of trajectories τ with high rewards $R(\tau)$. To update the policy parameters θ, REINFORCE uses gradient ascent:

$$\nabla J(\theta) = E_{\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t=1}^{T} r_t\right)$$

Motivation for actor-critic networks. REINFORCE suffers from high variability in the cumulative reward values ($\sum_{t=1}^{T} r_t$), resulting in instability and slow convergence during training [50]. Thus, we use actor-critic networks, that learns the policy distribution for action selection, while the critic network estimates the baseline, acting as the expected long-term reward under a certain state [36]. Next, we outline their forward pass and update mechanism.

The actor-critic network. The actor-critic network consists of an actor, which selects actions using a policy $\pi(a_t|s_t;\theta)$, and a critic, which estimates action values via $V^\pi(s_t;\theta_v)$. The value function is approximated with n-step returns and updated every t_{max} actions or at terminal states. Parallel actor-learners stabilize training by exploring independently. The actor outputs a policy distribution $\pi_\theta(s_t)$ using a softmax layer, while the critic predicts $V(s_t;\theta_v)$ through a linear layer. Our system trains 32 actor and critic networks asynchronously (as first presented in [39]), enhancing exploration by allowing actors to explore different parts of the environment independently. Different exploration policies in each actor-critic maximize diversity, stabilizing learning and reducing training time proportionally to the number of parallel actor-learners.

Training the actor critic networks. Recall that in REINFORCE, the high variance of cumulative rewards leads to unstable and inefficient training. To address this, we use the value function V as a baseline, and subtract it from the rewards to reduce the variance produced by the critic network. This has been shown to not introduce bias [50]. Specifically, we replace the cumulative reward $\sum_{t=1}^{T} r_t$ by the advantage function $A(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, where $Q^\pi(s_t, a_t)$ is the expected return starting from state s_t, taking action a_t, then following policy π. Hence, the gradient becomes:

$$\nabla J(\theta) \approx \sum_{t=0}^{T} [\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot A(s_t, a_t)]$$

where $A(s_t, a_t)$ can be estimated by $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$, known as the *temporal-difference* (TD) error. From another perspective,

since the baseline (i.e., the V value) represents the expected long-term reward of a state, the TD error reflects the advantages and disadvantages of different actions from that state.

We now discuss how to update the critic network. To estimate the V value accurately, the difference between $r_t + V^{\pi}(s_{t+1})$ and $V^{\pi}(s_t)$ should be minimized. Hence, the TD error can also be used to update the critic, and the loss function becomes:

$$L_{\phi} = \left(r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)\right)^2$$

Concretely, during each episode the policy and value function estimates are updated using the following rule:

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta') \cdot A(s_t, a_t; \theta, \theta_v)$$

where θ and θ_v represent the parameters for the policy and value function, respectively. $A(s_t, a_t; \theta, \theta_v)$ is an approximation of the advantage function, which quantifies the advantage of taking action a_t in state s_t.

Proximal update of the policy. We can further improve the quality of the selected tuples by updating the weights of the network more moderately. This is done by adjusting the loss function to include a proximal policy update, which creates a *trust region* by clipping the *surrogate objective*. This results in more moderate updates of the policy, as unconstrained maximization of the surrogate objective can lead to excessively large policy updates. The key mechanism is gradient clipping, which constrains updates to be within an ϵ range, preventing instability.

$$L^{\text{CLIP}}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

where $r_t(\theta)$ is the probability ratio, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. Maximizing the surrogate objective $\hat{E}_t[r_t(\theta)\hat{A}_t]$ without clipping can lead to unstable, overly large policy updates.

Further improvements. To encourage exploration, we incorporate the entropy of the policy π into the objective function by adding $\lambda\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s_t))$ to each step component. In this way, the agent is motivated to explore different actions and learn a more diverse and effective policy. In addition, to avoid selecting the same tuple multiple times, we implement *action masking* [18] for each algorithm. Action masking enforces a constraint that forbids the agents from selecting certain actions, specifically in our case those that would result in the repetition of a tuple.

To promote diversity among the selected tuples, we employ a regularization function that quantifies the degree of diversity in the chosen approximation set using a number in $[0, 1]$ which is added to the objective function. This approach is based on state of the art regularization techniques in [52].

Overall training algorithm. REINFORCE first initializes the parameters. In each iteration, the system selects a batch of tuples from different tables using the learned policy. Then it computes an estimate of the advantage function for each selected tuple to compute the loss and update the policy parameters. Finally, it outputs a trained actor-critic network, with an optimized policy of selecting tuples for approximating non-aggregated queries.

5.2 Transformation to RL Environment

Unlike computer games, where images and simple discrete information (such as remaining lives or earned points) serve as the main source of state, and actions are simple (such as going one unit in a given direction), the tabular domain is much more complex. As a result, RL has not been frequently used. Our work therefore focuses on constructing an optimal environment specifically designed for our use case and suitable for an RL model. In this section, we elaborate on the chosen environment (introduced in Section 4.2) and discuss alternative environments examined in ablation studies during our experiments (Section 6).

Environment	Agent	IMDB		MAS	
		Score	Total (m)	Score	Total (m)
GSL	HARLM	**0.64±0.06**	60 ± 7	**0.754± 0.025**	30 ± 2
GSL	HARLM- ppo	0.536 ± 0.003	47 ± 5.3	0.623 ± 0.071	23 ± 3
GSL	HARLM- ppo - ac	0.496 ± 0.002	72 ± 4.3	0.618 ± 0.021	44 ± 2.2
DRP	HARLM	0.365 ± 0.33	180 ± 12.3	0.455 ± 0.028	97 ± 3.5
DRP	HARLM- ppo	0.364 ± 0.004	165 ± 9.3	0.429 ± 0.001	83 ± 2.5
DRP	HARLM- ppo - ac	0.401 ± 0.016	203 ± 7.3	0.423 ± 0.011	82 ± 2
DRP + GSL	HARLM	0.569 ± 0.02	73 ± 5.1	0.619 ± 0.001	49 ± 3.6
DRP + GSL	HARLM- ppo	0.51 ± 0.01	71 ± 3.3	0.59 ± 0.024	43 ± 2.9
DRP + GSL	HARLM- ppo - ac	0.391±0.012	82 ± 5.3	0.51 ± 0.002	44 ± 3.5

Table 2: RL ablation study showing for each Environment and Agent the quality score and the setup total time (minutes)

Gradual-Set-Learning (GSL) environment. Our system employs a tailored *gradual-set-learning* (GSL) environment, which begins with an empty set. In each action, the system adds a predefined number of tuples from different tables, drawn from the action space generated during the pre-processing phase. The number of tuples added per action is parameterized to ensure efficiency and is typically set to a small, fixed value based on empirical performance, preventing an explosion in the action space. When tuples are selected from multiple tables, the system leverages join patterns from the query workload to guide the selection of tuples likely to be joined, further maintaining the relevance and manageability of the action space. After performing an action a_t and reaching a state S_{t+1}, the system evaluates the score of the new state using the metric described in Section 3, $Score(S_{t+1})$, which serves as the reward for the RL agent. An episode concludes either when a terminal state is reached or when the agent reaches k tuples (the k'th action).

Drop-One (DRP) Environment. The *drop-one* (DRP) environment provides an alternative setup. Each episode ends upon reaching a terminal state or a fixed horizon. Starting with a set of k tuples, the RL agent iteratively chooses to (1) remove a tuple, (2) add a new one, or (3) leave the set unchanged. After each action a_t, the updated state S_{t+1} (size k) is evaluated using the metric from Section 3, and the reward is $score(S_{t+1}) - score(S_t)$. Episodes typically last $100K$–$500K$ steps. DRP suffers from initialization instability and often gets stuck in suboptimal sets. Despite trying multiple initialization strategies, these issues persisted. Hybrid setups combining GSL and DRP (Section 6) also underperformed, showing greater deviation and less stability than GSL alone.

6 EXPERIMENTS

We now present results of a comprehensive evaluation of HARLM. We start by discussing the experimental setting (Section 6.1) before giving details of the experimental results (Section 6.2). Results of a user study and ablation studies are given in Sections 6.3 and 6.4.

As our experiments show, the approximation sets chosen by HARLM are consistently better than alternative approaches for both non-aggregate (see Table 1) and aggregate (see Figure 8) queries.

Baseline	Memory Size (k)										Frame size (F)			
	1000		5000		10000		15000		30000		25	50	75	100
	Time (m)	Score	Time (m)	Score	Time (m)	Score	Time (m)	Score	Time (m)	Score	Score	Score	Score	Score
HARLM	60 ± 7	**0.69±0.0425**	82 ± 3	**0.747 ±0.014**	107 ± 6	**0.795 ± 0.025**	127 ± 5.4	**0.864 ± 0.032**	162 ± 4.6	**0.978 ± 0.011**	**0.73 ± 0.325**	**0.69±0.043**	**0.631 ± 0.013**	**0.591 ± 0.024**
VAE	1920 ± 3.5	0.024 ± 0.0	1929 ± 2.1	0.024 ± 0.0	1941 ± 2.3	0.024 ± 0.001	1950 ± 1.8	0.036 ± 0.001	1959 ± 2.	0.071 ± 0.001	0.024 ± 0.0	0.024 ± 0.0	0.012 ± 0.001	0.012 ± 0.001
CACH	330 ± 12.5	0.152 ± 0.073	330 ± 12.5	0.20 ± 0.03	330 ± 12.5	0.239 ± 0.045	330 ± 12.5	0.254 ± 0.08	330 ± 12.5	0.44 ± 0.039	0.189 ± 0.01	0.172 ± 0.073	0.139 ± 0.062	0.1 ± 0.02
RAN	0.72 ± 0.08	0.246 ± 0.032	0.72 ± 0.08	0.435 ± 0.034	0.72 ± 0.08	0.469 ± 0.031	0.72 ± 0.08	0.501 ± 0.032	0.72 ± 0.08	0.64 ± 0.031	0.301 ± 0.036	0.246 ± 0.032	0.15 ± 0.041	0.11 ± 0.024
QUIK	160 ± 2	0.296 ± 0.036	174 ± 4	0.39 ± 0.021	189 ± 3.5	0.47 ± 0.022	206 ± 3	0.525 ± 0.051	224 ± 4.2	0.645 ± 0.072	0.315 ± 0.041	0.296 ± 0.036	0.175 ± 0.042	0.145 ± 0.031
VERD	200 ± 5	0.387 ± 0.034	208 ± 3	0.46 ± 0.082	223 ± 3	0.54 ± 0.021	234 ± 2	0.57 ± 0.017	254.5 ± 2.5	0.73 ± 0.034	0.34 ± 0.034	0.387 ± 0.034	0.26 ± 0.026	0.22 ± 0.051
SKY	1500 ± 1	0.340 ± 0.001	1872 ± 1	0.481 ± 0.001	2206 ± 1.1	0.551 ± 0.001	2604 ± 1	0.594 ± 0.0	N/A	N/A	0.446 ± 0.001	0.340 ± 0.001	0.247 ± 0.001	0.202 ± 0.001
BRT	2880	0.347 ± 0.004	2880	0.45 ± 0.003	2880	0.542 ± 0.07	2880	0.566 ± 0.003	N/A	0.67 ± 0.01	0.42 ± 0.003	0.347 ± 0.004	0.25 ± 0.014	0.22 ± 0.009
QRD	1800 ± 1	0.350 ± 0.045	1930 ± 2	0.316 ± 0.065	2160 ± 2.5	0.432 ± 0.02	2401 ± 4.1	0.512 ± 0.05	2863 ± 3	0.72 ± 0.042	0.368 ± 0.044	0.350 ± 0.045	0.303 ± 0.041	0.28 ± 0.061
TOP	338 ± 21	0.364 ± 0.001	338 ± 21	0.406 ± 0.001	338 ± 21	0.482 ± 0.001	338 ± 21	0.526 ± 0.001	338 ± 21	0.606 ± 0.0	0.401 ± 0.018	0.364 ± 0.001	0.257 ± 0.002	0.231 ± 0.001
GRE	2880	0.29 ± 0.01	2880	0.30 ± 0.02	2880	0.33 ± 0.015	2880	0.38 ± 0.021	N/A	N/A	0.29 ± 0.015	0.29 ± 0.01	0.17 ± 0.019	0.14 ± 0.03

Table 3: Quality for Different Memory and Frame Sizes

With a memory size of 15,000 tuples (approximately 0.1% of the data), our approach achieves 85% quality, while alternative approaches struggle to reach 60% (Table 3). Even under an extreme memory constraint of only 1000 tuples, HARLM maintains an average quality of 70%, surpassing baselines which struggle to achieve 40%. Our solution is also relatively efficient, generating a high-quality approximation set in just one hour compared to 30 − 40 hours required by some of the baselines (Table 1). This preprocessing time is acceptable for offline generation of the approximation set before data exploration sessions begin, akin to previous AQP works' offline processing. Our user study also validates the reasonableness of the wait time, considering the significant time savings during data exploration. The lighter version, HARLM-light (Section 4.4), creates the approximation set in just 30 minutes with a minor drop in quality.

6.1 Experimental Setting

Datasets. We evaluated the performance on the datasets: (1) *IMDB-JOB* [26] contains both data (entertainment domain, 34M tuples) and a query workload. (2) *MAS* [35] contains data (academic domain, 600K tuples). The query workload comes from [4]. (3) *FLIGHTS* [11] - contains flight delays information. The queries are generated according to [25]. (4) *TPC-H* [53] contains 5GB of data. The queries are generated by methods inspired by [17, 32, 45].

Baselines. To assess the effectiveness of HARLM, we conduct comparisons with various baselines designed to address related tasks and potentially offer candidate solutions for the HAQP problem (as detailed in Section 2). The baselines considered are categorized as naive, database-oriented, and generative models, with each category containing several different representatives. The execution of each baseline was restricted to 36 cores with 10GB each, across 500-1K processors, and with a 48 hours limit.

Naive baselines. (1) *Random sampling* (RAN) randomly selects rows from the large dataset. (2) *Brute Force* (BRT) exhaustively checks different combinations of k tuples to find the optimal solution. (3) *Greedy sampling* (GRE) selects, in each iteration, the row that achieves the largest marginal gain with respect to the metric, eliminates this row, and repeats. (4) *Top queried tuples* (TOP) selects the rows that appear most frequently in the results of the query workload (up to k tuples).

Baselines from database domains. (5) *Caching* (CACH) simulates a database cache [2] with limited memory by preserving tuples from the last executed query and evicting the least recently used (LRU) tuples to accommodate new ones.[2] (6) *Query result diversification* (QRD), based on [30], is an iterative approach that selects cluster medoids and reassigns data points to their nearest medoids. (7) *Skyline* (SKY) is a summarization method based on [43], extended to handle categorical data. (8) *VERDICT* (VERD) is an AQP method [45] that relies on sampling, query rewriting, and answer adjustment from samples. (9) *QUICKR* (QUIK) is another AQP method [23] that uses sampling and table statistics to maintain a catalog of plans and samples, selecting the appropriate samples at the right time.

Generative model baseline. (10) *Generative Model* (VAE) is a Variational Autoencoder method [51], a state-of-the-art generative model for AQP that generates fictitious tuples and executes queries over them.

Hyperparmeters. The actor network we use consists of an input layer followed by 2 fully-connected layers and a softmax layer for getting the logits for choosing the actions. The critic follows a similar architecture except the output is a single number. We use a learning rate of $5 \cdot 10^{-5}$, KL coefficient of 0.2 and entropy coefficient of 0.001. Finally, we set $k = 1000$ which is small enough to make the problem interesting but still large enough to make the chosen approximation set perform well for queries in the workload. Additionally, we set $F = 50$ which is close to real world values but still small enough to be informative for a human. Finally, by default the system executes all the queries given during training. Validation of all these values is given in Section 6.2 and 6.4.

Environment. HARLM is implemented in Python 3.9.13 and is an open source library [8]. It can therefore be used, e.g. in common Exploratory Data Analysis environments such as Jupyter notebooks, to load subsets of any database. We used Ray [38] for Reinforcement Learning, OpenAI's Gym [6] for the environment interface and Pytorch [46] for the models. The experiments were run on an Intel Xeon CPU- based server with 24 cores and 96 GB of RAM as well as 2 NVIDIA GeForce RTX 3090 GPUs.

Problem Justification. We conducted an experiment to illustrate the time required for querying the database directly, emphasizing the need for solutions handling both non-aggregate and aggregate queries. Using versions of the *IMDB* dataset of varying sizes, we averaged query execution times and accumulated results. As shown in Figure 6, even with a 1GB dataset, cumulative query time exceeds 5 hours after just seven queries. Limiting query results to 50-100 rows does not significantly reduce execution time, as

[2] In realistic scenarios, the order of query execution may not be neatly separated by user interests, as different users with diverse interests could query the same database simultaneously. In our experiments, we assume this realistic use case.

join and filtering processes remain computationally expensive. Figure 6 highlights these execution times, underscoring that lengthy exploratory queries lie beyond typical database optimizations, as discussed in Section 1.

6.2 Overall Evaluation

We compare our system to baselines on quality and running time across various datasets. The workload Q is split into training (Q_{train}) and test (Q_{test}) sets. Baselines use Q_{train} as needed, and evaluation is conducted over Q_{test} using the quality metric from Section 3 (Equation 1) for SPJ queries, and relative error for aggregate queries [17, 45, 51]. Results are averaged across multiple train-test partitions, with variance reported. All experiments follow the system's default parameters unless noted otherwise, with deviations examined via ablation studies.

Non-Aggregate Queries Quality Assessment. Figure 1 presents a quality comparison across two datasets (*IMDB* and *MAS*), reporting the average quality score and setup time in minutes. All baselines were capped at 48 hours. GRE and BRT failed to finish in time despite parallelization, and we report their best observed scores. As shown, our method outperforms others even under a memory constraint of k=1000, which corresponds to less than 0.0001% of *IMDB* and 0.001% of *MAS*.

Performance variance is attributed to dataset and workload complexity. Naive baselines like BRT and TOP underperform on larger workloads. VAE produces low scores due to its inability to generate tuples satisfying query filters or generalize across the workload, highlighting the difficulty of learning approximation sets for selection queries. Performance improves with increased memory, along with better running times (discussed later). Section 6.4 shows quality trends under different parameters. We also report results for HARLM-light, a lightweight version of HARLM trained on 25% of the data with a learning rate of 0.1, which achieves slightly lower scores.

Query execution time after approximation set creation is consistent across baselines, as all sets are the same size. On average, answering 10 queries takes 0.16±0.1 minutes for *IMDB* and 0.1±0.12 for *MAS*. The exception is VAE, which requires online generation and evaluation, taking 5 ± 0.03 and 2.5 ± 0.22 minutes, respectively. We therefore focus evaluation on approximation set quality and setup time.

Aggregate Queries Quality Assessment. We compared our solution to recent leading approaches: GAQP [51], DEEPDB [17] and DBEST++ [31]. GAQP uses *Variational Autoencoders* to generate tuples and execute AQP queries on the generated data. DEEPDB, on the other hand employs *Sum-Product Networks* to estimate the query answers. Finally, DBEST++ uses a mixture of models to estimate individual columns and relationships between pairs of columns. We used the *FLIGHTS* dataset and a workload of 1000 aggregate queries generated by [11]. These queries were divided into train-test sets for evaluation. For GAQP, we recreated the experiments on the *FLIGHTS* and the *TPC-H* [53] datasets mentioned in [51], using a memory size of 1%. To evaluate the results, we employed a common metric for AQP tasks known as relative error (used in both [51] and [17]). This metric compares the predicted answer (a_{pred}) to the true answer (a_{truth}) of the query, and is defined as follows: relative error = $\frac{|a_{pred} - a_{truth}|}{|a_{truth}|}$. For group-by queries, the relative error is computed for each group individually and then averaged. In cases where there are missing groups in the output, the relative error is set to 1 to indicate a complete mismatch between the predicted and true answers. This metric allows us to quantitatively assess the accuracy of our system's predictions in comparison to the ground truth values.

Baseline	% c_ins	#no_c_ins	#ins	easiness	rushing	successful	hardness	irritation
HARLM	**75±3**	**0**	**10±2**	**4.0±0.3**	**2.7±0.8**	**4.5±0.2**	**1.9±0.4**	**1.1±0.8**
DEFAULT	40 ± 5	2	5 ± 1	1.8 ± 0.2	4.2 ± 0.8	1.5 ± 0.1	3.5 ± 0.4	4.0 ± 0.2

Table 4: User-Study results

In Figure 8, we present the relative errors for different query categories, including queries with sum, average, and count operators, both with (G+SUM, G+AVG, G+CNT) and without (SUM, AVG, CNT) group by clauses. Notably, none of the existing approaches outperforms our system across all operators. In half of the operators, our system attains the lowest error rate, surpassing state-of-the-art approaches, while in the remaining cases, we exhibit comparable performance to at least one baseline.

Efficiency. We compare the system's running time with other methods, focusing on *set-up* (*setup*) time, the time (in minutes) the method requires to create the approximation set (Table 1). HARLM consistently shows good performance, with a lower setup time than many of the baselines. Although naive baselines like Random (RAN) have a lower setup time, their quality, as indicated in the *Score* column, is notably poorer. Note that HARLM-light, shows a remarkable improvement in the setup time of the system, requiring only 32 minutes as opposed to 60 minutes for the *IMDB* dataset.

Estimator Quality. A pivotal aspect of our system during the inference phase is the estimator, which, when presented a user query, gauges whether the constructed approximation set (Section 4.3) can effectively answer the query. To evaluate this estimator, we conduct a series of experiments, shown in Figure 7. These experiments involved acquiring a set of test and training queries, generating the approximation set using our system, and subsequently querying the estimator about the answerability of each test query. This information is then juxtaposed with the scores assigned to the queries over the approximation set using Equation 1. Adopting a threshold of 0.5 and above as "answerable," we calculated recall and precision to quantitatively compare the estimator's responses. As shown in the figure, our approach achieves a remarkable 0.95 recall and 0.90 precision. Further iterations of this process progressively limited the size of the training set (queries). As anticipated, this impacted the estimator's performance on the test set queries. Nevertheless, even with reduced training queries (50% utilization), the precision and recall remained high (0.75 and 0.85).

Building upon our estimator, we implemented the full version of our algorithm. In this variant, the actual database is queried whenever the estimator predicts a query to be unanswerable, allowing users the flexibility to decide on a per-query basis whether they are willing to endure the requisite time for a complete answer. The first test, which queries the database for predictions falling below 60% (according to Equation 1), attains an average score of 85% in contrast

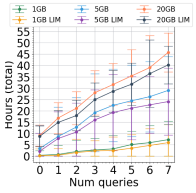

Figure 6: Query times

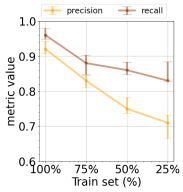

Figure 7: Estimator quality

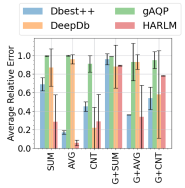

Figure 8a: Flights AQP

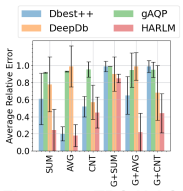

Figure 8b: TPC-H AQP

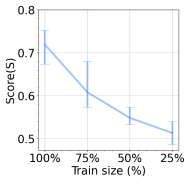

Figure 9a: Training set size

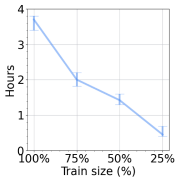

Figure 9b: Training set size

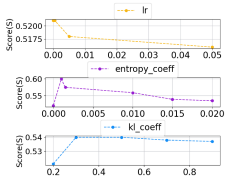

Figure 10: RL parameter tuning

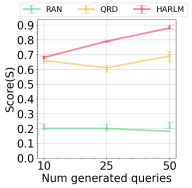

Figure 11: Unknown workload

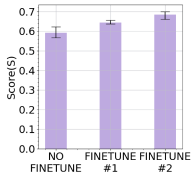

Figure 12: Finetuning the model

to the average of 70% shown in Table 1. As expected, this results in a rise in "QueryAvg" time to approximately 24 minutes due to querying the database. In a second test the database is queried for predictions below 80% and the average score is 76%, with about a 5-minute increase in query response time.

Unknown Query Workload. To address scenarios where the query workload is not initially available, we evaluated the system's performance using the *FLIGHTS* dataset, and some exploration queries written by data analysts (Figure 11). The system begins by generating several queries, while also allowing the user to contribute a subset of queries to refine the generated set. After integrating the user-provided queries and constructing an approximation set, we evaluated the quality of the approximated answers. As the user continues to submit additional queries, the system further fine-tunes its model, leading to improved performance. In this experiment, the user added five queries at each step, which helped the system generate queries that better aligned with the user's interests, continuously fine-tuning the model with more relevant queries. The results show a significant improvement in answer quality, starting at 70% for 10 queries and reaching up to 90% for 50 queries, outperforming alternative methods. We compared our approach against RAN and QRD, both of which can function without a predefined workload. While RAN does not consider the data or workload, QRD leverages inherent data patterns. However, QRD consistently underperformed compared to our method, with its scores never exceeding 70%.

Effect of Fine-Tuning. As detailed in Section 4.3, our system implements fine-tuning when it detects *interest drift* in user queries, which is managed using an estimator. To simulate this drift, we divide our workload into three distinct sub-tasks via a clustering algorithm applied to the embedded version of the queries, ensuring that the introduction of new queries triggers *interest drift*. We then select a test sample from each cluster for querying the system. Initially, the system is trained on the first cluster and tested with its corresponding workload. Gradually, we introduce parts of the test set containing unseen queries that differ from the original training workload. As expected, the estimator identifies these queries as unanswerable, which initiates the fine-tuning process using the training set of the second cluster. This process repeats as we incorporate the remaining training sets. The results (Figure 12), show a rapid improvement in the quality of the approximation set after fine-tuning, performance increasing from 60% to 70%. This experiment highlights the robustness of our system in handling new, unseen queries, a crucial aspect for exploration scenarios. It demonstrates HARLM's ability to adapt to evolving query workloads, ensuring reliable generalization to unexpected query types.

Diversity. We evaluated the diversity of query answers generated by our system compared to baselines. Although we considered incorporating a diversity factor into the scoring function, it was omitted due to performance trade-offs. Using pairwise *Jaccard distance* as a standard metric, we conducted experiments on the *IMDB* dataset, inspired by prior diversity metrics (e.g., [49, 56]). For queries with LIMIT 100, database results had 58% average diversity, while our solution achieved 52%, at least 14% higher than most baselines except RAN. Notably, while RAN matches our diversity, it performs poorly in approximation quality.

6.3 User Study

To subjectively verify our system's effectiveness, we followed the approach of prior AQP studies ([13, 37, 47]) and conducted two user studies to assess whether the system accelerates the exploration process, aids in gaining better insights, and garners user acceptance.

Motivation and Quality Metric Verification. The first study aimed to validate the motivation behind our problem, suggesting that users typically focus on the initial rows in the result set for non-aggregate queries. It also aimed to assess the quality of the approximation created by our system, which is indistinguishable from querying the entire database. Twenty users familiar with the dataset ([35]) were presented with a set of SQL queries. For each query, they were asked to identify which answer was generated by our model and which was the database's answer. Users found the two answers indistinguishable, confirming that not all rows are necessary in the response. We constrained users from writing their own queries to ensure comparability between participants.

Exploration Task Effectiveness. In the second study, 12 participants engaged in a more immersive exploration with our system and were tasked with time-limited exercises using the dataset in [26] (available at [8]). There, participants were instructed to formulate their own queries and document insights gained, akin to the methodology outlined in [3]. They were divided into two groups: one queried the full database, and the other queried a subset generated by our system (users did not know what group they were in). Their insights were subsequently validated using a ground truth source available at [8]. As shown in Table 4 (left columns), users of HARLM generated a significantly higher number of queries, facilitated by the reduced query time, enabling faster data comprehension and progression to subsequent queries. Furthermore, users of HARLM attained a greater number of correct insights compared to those querying the database, with none of the former group reporting zero correct insights. Upon completion, participants rated the difficulty and success of their experience using the *NASA Load Index* [16]. The findings demonstrate that HARLM's users experienced reduced time pressure and frustration, and perceived greater success (Table 4, right columns).

6.4 Ablation Studies and Parameter Optimization

Effect of Memory Size Constraint (k) on Quality. We evaluated the impact of memory size (k) on performance and setup time, testing values of k in the range $[10^3, 5 \cdot 10^3, 10 \cdot 10^3, 15 \cdot 10^3, 30 \cdot 10^3]$. As expected, performance improved across all approaches as the approximation set size increased. For some baselines, setup time also grew with larger k. Table 3 shows that our method consistently outperforms baselines, achieving an average score of 80% at $k = 15 \cdot 10^3$ and an impressive 97% at $k = 30 \cdot 10^3$. This is double the score of GRE and 20% higher than SKY or QRD. That supports our default value as a balance between performance and runtime. The baselines failed to complete within the time limits marked N/A.

Effect of Frame Size Constraint (F) on Quality. We also experimented with varying the *frame size* (F) within the range [25, 50, 75, 100]. As the frame size increases while keeping memory size constant, the problem becomes more challenging, requiring more tuples per query, which leads to a general decline in quality (with no significant change in the setup time). Nevertheless, HARLM consistently outperforms other baselines, as illustrated in Table 3. Its performance remains relatively stable compared to, for instance, SKY, which drops from 40% to 20%. Notably, an optimal value for F is typically below a few dozen tuples.

Effect of Training Set Size on Quality and Time. As mentioned in Section 4.1, our system takes a set of training queries and uses an embedding model to determine the most significant queries to execute. This improves the overall training time, since running queries is time-consuming. However, it comes at the cost of quality since exposing the system to fewer queries may impact its ability to accurately answer similar queries from the test set when training concludes. In Figure 9a, we observe the change in quality as the size of the training set $\hat{Q}_{\text{train}}$ decreases (as is seen in the percentages in the x-axis), with our method maintaining reasonable quality compared to other baselines (as seen in Figure 2). Additionally, in Figure 9b, we see the training time for the same set of experiments, with training time decreasing to approximately 30 minutes.

RL Hyper-parameter Tuning. We examined the effect of tuning the key hyper-parameters of our RL algorithm, and present the quality results here. Figure 10 illustrates the impact of adjusting the learning rate (lr) across the range $[5 \cdot 10^{-5}, 5 \cdot 10^{-4}, 5 \cdot 10^{-3}, 5 \cdot 10^{-2}]$, the entropy coefficient in the range $[0, 0.001, 0.0015, 0.01, 0.015, 0.02]$, and the KL coefficient across $[0.2, 0.3, 0.5, 0.7, 0.9]$. We found that the entropy coefficient significantly influences the algorithm's success, and we set it to 0.001. The KL coefficient, though important, has a somewhat smaller effect, and we set it to 0.3. As is common in learning algorithms, higher learning rates can lead to faster convergence, but often at the cost of model stability.

Ablation Studies. Table 2 summarizes ablation studies conducted on our RL architecture, aiming to validate the contribution of each component to the overall system performance. Different environments, detailed in Section 5 and referred to in the Environment column, were examined, specifically crafted to transform our problem into a tabular RL environment. For each environment, we assessed the impact of removing individual components from our agent, such as the actor-critic and loss clipping (-ac and -ppo in the Agent column). The results indicate that the GSL environment is most suited for our use case. Moreover, the use of actor-critic and clip loss is crucial for high performance.

7 CONCLUSIONS AND FUTURE DIRECTIONS

We introduced the problem of approximating answers to non-aggregate queries and proposed a solution for approximating both aggregate and non-aggregates queries. Our system, HARLM, generates an approximation set, a subset of the dataset, over which queries can be executed to deliver fast, high-quality results. Since an exact solution is impractical, we leverage RL to provide a robust approximate solution. HARLM overcomes various challenges, including the size of the action-space and the necessity to generalize beyond the known workload. Experiments demonstrate HARLM's efficacy compared to other baselines for aggregates and non-aggregates queries. Our future work will focus on devising approximation sets with problem-specific constraints, including fairness aspects.

ACKNOWLEDGMENTS

The research was supported by ISF - the Israel Science foundation - grant 2707/22 of the Breakthrough Research Grant (BRG) Program.

REFERENCES

[1] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*. 29–42.

[2] Munir Ahmad, Muhammad Abdul Qadir, and Muhammad Sanaullah. 2008. Query processing over relational databases with semantic cache: A survey. In *2008 IEEE International Multitopic Conference*. IEEE, 558–564.

[3] Yael Amsterdamer, Susan B Davidson, Tova Milo, Kathy Razmadze, and Amit Somech. 2023. Selecting Sub-tables for Data Exploration. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2496–2509.

[4] Dana Arad, Daniel Deutch, and Nave Frost. [n.d.]. LearnShapley: Learning to Predict Rankings of Facts Contribution Based on Query Logs. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 4788–4792.

[5] Ori Bar El, Tova Milo, and Amit Somech. 2020. Automatically Generating Data Exploration Sessions Using Deep Reinforcement Learning *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3318464.3389779

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[7] Graham Cormode. 2017. Data sketching. *Commun. ACM* 60, 9 (2017), 48–55.

[8] Susan B. Davidson, Tova Milo, Kathy Razmadze, and Gal Zeevi. 2024. *HARLM Github Repository*. https://github.com/GalZeevi/HARLM

[9] W Edwards Deming and Frederick F Stephan. 1940. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *The Annals of Mathematical Statistics* 11, 4 (1940), 427–444.

[10] K Dhanasree and C Shobabindu. 2016. A survey on OLAP. In *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*. IEEE, 1–9.

[11] Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2020. Idebench: A benchmark for interactive data exploration. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1555–1569.

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

[13] Danyel Fisher, Igor Popov, Steven Drucker, and m.c. schraefel. 2012. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 1673–1682. https://doi.org/10.1145/2207676.2208294

[14] Terry Gaasterland. 1997. Cooperative answering through controlled query relaxation. *IEEE Expert* 12, 5 (1997), 48–59.

[15] Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2017. Revisiting Reuse for Approximate Query Processing. *Proc. VLDB Endow.* 10, 10 (June 2017), 1142–1153. https://doi.org/10.14778/3115404.3115418

[16] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.

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

[18] Shengyi Huang and Santiago Ontañón. 2020. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171* (2020).

[19] IBM. [n.d.]. https://www.ibm.com/topics/tm1

[20] IBM. [n.d.]. https://www.ibm.com/docs/sr/planning-analytics/2.0.0?topic=tier-tm1-server-installation

[21] Martin Idel. 2016. A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps. arXiv:1609.06349 [math.RA] https://arxiv.org/abs/1609.06349

[22] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[23] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *Proceedings of the 2016 international conference on management of data*. 631–646.

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

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

[26] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.

[27] Carson K Leung, Yubo Chen, Calvin SH Hoi, Siyuan Shang, and Alfredo Cuzzocrea. 2020. Machine learning and OLAP on big COVID-19 data. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 5118–5127.

[28] 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 (2018), 379–397.

[29] B. Liu and H. V. Jagadish. 2009. Using Trees to Depict a Forest. *PVLDB* 2 (2009), 133–144.

[30] Bin Liu and H. V. Jagadish. 2009. Using Trees to Depict a Forest. *Proc. VLDB Endow.* 2, 1 (aug 2009), 133–144. https://doi.org/10.14778/1687627.1687643

[31] Qingzhi Ma, Ali M Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, and Peter Triantafillou. 2021. Learned approximate query processing: Make it light, accurate and fast. In *Conference on Innovative Data Systems,(CIDR21)*.

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

[33] Imene Mami and Zohra Bellahsene. 2012. A survey of view selection methods. *Acm Sigmod Record* 41, 1 (2012), 20–29.

[34] Pasin Manurangsi. 2018. A Note on Max k-Vertex Cover: Faster FPT-AS, Smaller Approximate Kernel and Improved Approximation. *arXiv preprint arXiv:1810.03792* (2018).

[35] Microsoft. 2016. *Microsoft academic search*. http://academic.research.microsoft.com

[36] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[37] 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*. Association for Computing Machinery, 2904–2915. https://doi.org/10.1145/3025453.3025456

[38] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 561–577.

[39] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. 2015. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296* (2015).

[40] Adhish Nanda, Swati Gupta, and Meenu Vijrania. 2019. A comprehensive survey of OLAP: recent trends. In *2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, 425–430.

[41] Laurel Orr, Magdalena Balazinska, and Dan Suciu. 2020. Sample debiasing in the themis open world database system. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 257–268.

[42] Laurel J Orr, Samuel K Ainsworth, Kevin G Jamieson, Walter Cai, Magdalena Balazinska, and Dan Suciu. 2020. Mosaic: A Sample-Based Database System for Open World Query Processing. In *CIDR*.

[43] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive Skyline Computation in Database Systems. *ACM Trans. Database Syst.* 30, 1 (mar 2005), 41–82. https://doi.org/10.1145/1061318.1061320

[44] Y. Park, M. Cafarella, and B. Mozafari. 2016. Visualization-aware sampling for very large databases. In *ICDE*.

[45] 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*. 1461–1476.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[47] Sajjadur Rahman, Maryam Aliakbarpour, Ha Kyung Kong, Eric Blais, Karrie Karahalios, Aditya Parameswaran, and Ronitt Rubinfield. 2017. I've seen enough: Incrementally improving visualizations to support rapid decision making. 10, 11 (1 Aug. 2017), 1262–1273. https://doi.org/10.14778/3137628.3137637

[48] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[49] Mariia Seleznova, Behrooz Omidvar-Tehrani, Sihem Amer-Yahia, and Eric Simon. 2020. Guided exploration of user groups. *PVLDB* 13, 9 (2020), 1469–1482.

[50] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[51] Saravanan Thirumuruganathan, Shohedul Hasan, Nick Koudas, and Gautam Das. 2020. Approximate query processing for data exploration using deep generative models. In *2020 IEEE 36th international conference on data engineering (ICDE)*.

IEEE, 1309–1320.
[52] Yingjie Tian and Yuqi Zhang. 2022. A comprehensive survey on regularization strategies in machine learning. *Information Fusion* 80 (2022), 146–166.
[53] Transaction Processing Performance Council. [n.d.]. TPC Benchmark H (Decision Support). http://www.tpc.org/tpch/. Accessed: 2025-04-04.
[54] Muhammad Habib ur Rehman, Chee Sun Liew, Assad Abbas, Prem Prakash Jayaraman, Teh Ying Wah, and Samee U Khan. 2016. Big data reduction methods: a survey. *Data Science and Engineering* 1 (2016), 265–284.
[55] Marcos R Vieira, Humberto L Razente, Maria CN Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina, and Vassilis J Tsotras. 2011. On query result diversification. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 1163–1174.
[56] Ting Wu, Lei Chen, Pan Hui, Chen Jason Zhang, and Weikai Li. 2015. Hear the Whole Story: Towards the Diversity of Opinion in Crowdsourcing Markets. *PVLDB* 8, 5 (2015).