HARLM: Holistic Approximation RL Model

Anonymous Author(s)

ABSTRACT

In data exploration, executing queries over large datasets can be time-consuming. Previous work has proposed approximate query processing as a way to speed up aggregate queries in this context, but neglected non-aggregate ones. Our paper introduces a novel holistic approach to handle both types of queries, focusing on 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. This approach 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).

1 INTRODUCTION

During the data exploration phase of data science, users frequently query iteratively to better understand the data [8]. 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) [29], 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 data exploration phase.

Example 1.1. In a study of IMDB exploratory data analysis (EDA) sessions [27], approximately 50% of the queries were complex SPJ queries. To illustrate, consider a query that explores the connection between production companies' origin country and movie cast size, involving a complex query across five tables with various filters. These queries are iteratively refined as analysts understand relevant data subsets through table and column exploration, and the refinement process includes both aggregate and non-aggregate queries. Despite limiting the result size, processing these non-aggregate queries take a substantial amount of time: On average, they required 25 minutes to execute, with 30% surpassing the 1-hour mark. Note that lengthy exploratory queries are not uncommon, as they lie beyond the scope of the typical query workload for which the database has been optimized. Therefore, as in AQP, prioritizing quick responses in the form of a brief data overview (crucial for analysts [4]) rather than waiting for the full result, is reasonable. □

We 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 tables in the database, 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 exceptionally large results, the approximation can provide the user with a query answer limited to the frame size. We describe in greater detail the inadequacy of existing approaches (e.g. AQP methods, OLAP and data summarization and sampling methods) in 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 query 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 query workload. (C5) Detecting when there is a drift in user interest.

Note that computing approximation sets is time consuming, but can be conducted offline. Analogous to the trade-off identified in Approximate Query Processing (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 pre-processing the database (which may encompass multiple tables) and query workload to create a reduced subset of the data (C2). 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 a reinforcement learning model utilizing critic-actor networks with Proximal Policy Optimization (PPO), we specifically tailored for tabular data. The model encompasses a well-crafted environment and transformed action definitions (C1, C4). A specialized reward function aligns with the goal of selecting rows for accurate approximations (C3). This combination yields a robust, adaptive solution that significantly improves the efficiency and accuracy of non-aggregate query approximations. HARLM also detects drift in user interests, enabling model fine-tuning for better approximate results (C5).

Since the computation of approximation sets in HARLM can be time-consuming (in Section 6 we show that it can take up to an hour), 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, enhancing flexibility across diverse environments. (3) Flexible training within time constraints, providing estimated guarantees and 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, which contribute to overall system efficiency. (5) Incorporating 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. This is achieved through the generation of initial query workloads and dynamic model fine-tuning with adapted queries. These benefits position HARLM as a comprehensive tool for efficient and effective 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. Specific contributions include:

(1) **Problem Definition and Metric Introduction** (Section 3): We define the problem of approximating non-aggregate queries and introduce a novel metric for evaluating it's quality.
(2) **Holistic approximation RL-Based** (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. In AQP, two approaches are taken ([29]): (1) Retaining a small number of tuples with metadata, and (2) generating new tuples using generative models. Examples of the first approach include sampling methods [2, 10, 17], which expedite query processing by reducing tuples at the cost of exactness. While those methods are less effective for non-aggregate queries (Section 6), we draw inspiration from [44] to generate an initial approximation set using the given query workload, a basis for training a model that produces a high-quality approximation set. Examples of the second approach use generative models, such as generative adversarial networks (GANs) and Variational Autoencoders (VAEs), to generate synthetic or representative samples in the context of tabular data [14, 50]. These models offer advantages like explicit probabilistic modeling and capturing complex data distributions. However, generated tuples may significantly deviate from real tuples, leading to issues where user queries produce minimal or no results, return nonsensical outcomes, or even mislead the user (see Section 6), making them unsuitable.

OLAP. In Online Analytical Processing (OLAP), and especially Multidimensional OLAP (MOLAP), the focus is on aggregate operators managed through data cubes relying on past query workloads ([11, 41]). However, MOLAP poses several challenges: 1) the storage requirement for data cubes is substantial, about ten times the original dataset size [28]; 2) Implementing MOLAP systems requires DBAs with specific OLAP syntax expertise, limiting their use for typical data scientists [21]; 3) The setup process is intricate, and can take several days [22]. These challenges limit their accessibility within the data science community.

Data Summarization and Sampling. These techniques provide essential tools for various data processing tasks. Cluster-based representative selection aims to maximize data diversity by selecting instances from clustered data [30]. In visualization, row sampling reduces data points while minimizing error [43]. Query result diversification selects rows with both relevance and diversity, often utilizing greedy algorithms [53]. However, as demonstrated in Section 6, these approaches can incur a high running time and are optimized for different objectives, making them unable to provide high quality answers in our query approximation setup.

Data Reduction, Sketches and others. Data sketches and reduction techniques (e.g. [10, 52]), are important for data processing tasks. However, they may lose critical information needed for accurate selection query results, making them unsuitable for non-aggregate queries. Approaches like the skyline operator [42], caching operations [3], and view selection materialization with space constraints ([34]) prove to be inferior when tackling the combined problem of non-aggregate queries as well as aggregates (see Section 6).

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 $\mathcal{T} = \{T_1, ..., T_n\}$ and an initial query workload Q_I consisting of both aggregate and non-aggregate queries over $\mathcal{T}$. Each aggregate 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. We denote the transformed query workload by Q, and capture the importance of each query as $w : Q \rightarrow [0, 1]$, where $\sum_{q \in Q} w(q) = 1$.

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 to is to consider all subsets $S \subseteq T$ of size k, and find the one that maximizes $score(S)$. This is obviously computationally intractable. Worse yet, this naive solution cannot be significantly improved since the HAQP problem is NP-hard. This can be shown by reduction to the max-k-vertex cover problem [35]. 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 formulation of the 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 combinatorially even worse, as all combinations of subsets must be considered.

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). Lastly, 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).

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 address some of the challenges mentioned earlier, and provide the 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 before creating their vector representations. For this, we use existing *query*

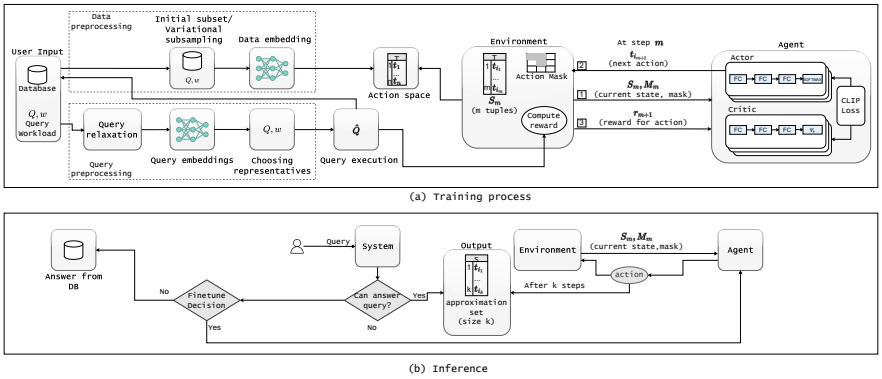


Figure 1: HARLM architecture.

relaxation methods [16], which loosen or modify query conditions when strict adherence to the original conditions do not yield sufficient results. In our setting, these modifications enlarge the result set for each query, thereby making the query more general. This also adds in tuples that are beyond those returned by the current workload, helping to generalize for future, not yet known, queries. Vector representations of the generalized queries are then created using a modified version of sentence-BERT [47]. We adapt the transformer-based model to capture similar queries by embedding queries into a vector space. Finally, we use a clustering algorithm over the embedded queries to select what we call the *query representatives* to be used in the score function.

Data Pre-processing. Data pre-processing starts by removing data that does not appear in the output of any query representative. Then it is further reduced using subsampling in order to create the action space to be used for training the model. This reduced subset of tuples (the action space) is then transformed into a vector representation to be ingested by the RL model and to ease the learning. The subsampling technique used for reducing the results of the query representatives is *variational subsampling* [44], using probabilistic models to approximate intractable likelihoods by introducing latent variables and optimizing their variational parameters. The transformation into a vector representation is achieved using another modified version of sentence-BERT*.

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 ([24, 44]). In our approach this is addressed during the data pre-processing phase, which creates a smaller set of tuples based on the generalized query workload, which may contain joins. The second challenge, $C2$, is the cost of executing queries in the query 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

The RL formulation of our problem treats it as a sequential decision-making process. The RL components during the training phase of ASQP-RL, depicted in Figure 1(a), and defined formally in Algorithm 1, are defined as follows.

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, in our case is represented as the set of actions previously selected by the model. Recall that an action, in this context, 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. In this report, we also discuss 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

*This version is designed to be more suitable for tabular rows, incorporating modifications such as including column names as tokens to capture both the meaning of the column as well as the value.

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 [37]. This method incorporates several actor networks and a critic network 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 each agent's component.

Overall training process. The training process, shown in Figure 1(a), takes as input the database D and the query workload Q and pre-processes them. In this phase, queries from the training workload are generalized and prepared for evaluation. The results of these modified queries are then used to construct the action space A. The training step begins with the iterative selection of the approximation set using the RL model. Each epoch of the training process processes batches of queries retained from the pre-processing stage, evaluating the chosen action and subsequently updating the model. Specifically, each state s_k represents the state after k epochs of the model, indicating that k actions were previously selected from the action space A. The agent, represented by the actor network, takes s_k as input and computes the probability of each action, denoted as $\pi_\theta(a|s) = p(a|s, \theta)$, where $a \in A$. A higher probability indicates a larger expected long-term reward for the corresponding action. This implies that the final approximation set is more likely to satisfy the query workload the model was trained upon. Multiple actor-critic networks operate asynchronously during this process to expedite training by considering more policies. Given the fact that actions represent tuples, we use action masking [20] to enforce that a tuple may only enter the approximation set once. In this technique, the environment provides the model along with the state a mask that shows the valid actions the model can choose from. The critic network estimates the long-term reward of the current state, and computes the loss error with the returned reward. The networks are then updated for optimized action selection. For more details 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 1 ASQP-RL Training

Input: Database D, query workload Q
Output: A trained RL model

$vec\_generalized\_Q = Emb\_sql(relaxation(Q))$
$\hat{Q} = rep\_selection(vec\_generalized\_Q)$
$\hat{D} = \hat{Q}(D)$
$initial\_set = Emb\_tab(variational\_subsampling(\hat{D}))$
for each *episode* during training **do**
    $batch\_Q = batch(\hat{Q})$
    action a_k is sampled based on $p(a_k|s_k, \theta)$, $a_k \in A$
    $s_{k+1} = s_k$ union a_k
    Reward r is then computed based on $state$ and a_k matching to $batch\_Q$
    Use the reward to calculate loss and update the network
    if early stopping (loss) **then**
        break
    end if
end for
return M

4.3 Inference and User's Interaction

The inference phase occurs after training the RL model (Figure 1(b)), and Algorithm 2. 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. After the initial approximation set is obtained, the user can issue both aggregate and non-aggregate queries. For each such query, the system estimates whether the model is likely to contain relevant tuples. This estimation is based on the closeness of the query (or the non-aggregate version of the query) to the training workload seen by the model, using the aforementioned query embeddings and the existing model's performance on the training workload. If the system deems the query answerable, it provides the answer derived from the approximation set. Otherwise, it queries the original database. As will be seen in Section 6, our system attains high accuracy in predicting whether a query is answerable. This accuracy significantly contributes to the overall precision of the results presented to the user, minimizing the likelihood of displaying partial results and potentially introducing bias into their analysis.

Identifying Interest Drift. Interest drift is identified when user queries deviate from the initial model training query workload. When three or more queries deviate from the training workload with confidence scores surpassing 0.8, our model initiates a fine-tuning process tailored to the specific characteristics of these queries, addressing $C5$. The parameters outlined above that are used to trigger this process is a deliberate effort to strike a balance between optimizing system performance and minimizing running time costs associated with fine-tuning, as detailed in Section 6.

Algorithm 2 ASQP-RL Inference

Input: Database D, trained model M, req_size- optional
Output: Approximation Set S

```
while |S| < req_size do
    Action a_k is sampled based on p(a|s_k, θ), a ∈ A
    // p(a|s_k, θ) is output by the model's M policy π
    append the tuples of a_k to S
end while
return S
```

4.4 Further Improvements

HARLM-light and adaptive configuration. We explored several enhancements to HARLM, and incorporated them in a version called HARLM-light. These include implementing an early-stopping threshold, increasing the learning rate, and reducing the number of queries considered during the training's pre-processing phase. Despite a slight loss in quality (about 10%), the running time is significantly reduced (to 30 minutes), as demonstrated in Section 6. At the same time, HARLM-light maintains superior performance compared to competitors. By factoring in user time constraints and the portion of the training query workload to be used as input, our system discerns the best configuration for the given use case. This involves adjusting various parameters, ranging from the lightest version to our highest-performing configuration in terms of quality. This adaptive methodology optimizes the trade-off between running time and the quality of the approximation set, tailoring our system to meet the requirements of users.

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.

Unknown Query Workloads. In Section 3, we define the problem as optimizing a metric function with respect to a given workload. To address scenarios where the query workload is unknown, we draw inspiration from works which use generated query workloads [25, 33]. Our system uses statistical information collected from the tables, such as the mean and standard deviation of numerical columns, a sampled set of categorical columns (with repetition to account for popularity of certain values), and standard query templates, to generate query workloads. While the query workload generated this way is adequate, as demonstrated in Section 6, we conduct additional query generation alongside user input queries during system interactions to ensure alignment with user interests. During the user's queries on the resulting approximation set, there is an iterative process of generating new queries that align with the user's interests and possibly fine-tuning the system. This incorporates both the user's queries and the newly generated queries, aiming for better alignment with user preferences.

Baseline	IMDB			MAS		
	Score	setup(m)	QueryAvg(m)	Score	setup(m)	QueryAvg(m)
HARLM	**0.64±0.06**	60 ± 7	0.16 ± 0.1	**0.75425 ±0.025**	30 ± 2	0.1 ± 0.12
HARLM-light	0.53±0.09	32 ± 2	0.16 ± 0.1	0.61 ±0.04	15 ± 0.3	0.1 ± 0.12
VAE	0.0025±0.002	1920 ± 3.5	5 ± 0.03	0.045 ± 0.001	720 ± 3	2.5 ± 0.22
CACH	0.084±0.06	330 ± 12.5	0.16 ± 0.1	0.2207 ±0.085	34 ± 1.2	0.1 ± 0.12
RAN	0.29±0.03	0.72 ± 0.08	0.16 ± 0.1	0.20275 ±0.034	0.68 ± 0.024	0.1 ± 0.12
QUIK	0.343±0.04	160 ± 2	0.16 ± 0.1	0.25025 ±0.0317	60 ± 3.1	0.1 ± 0.12
VERD	0.471±0.021	200 ± 5	0.16 ± 0.1	0.3045 ±0.0467	90 ± 2.3	0.1 ± 0.12
SKY	0.347±0.001	1500 ± 1	0.16 ± 0.1	0.33362 ± 0.001	480 ± 3	0.1 ± 0.12
BRT	0.297±0.007	2880 ± 0.	0.16 ± 0.1	0.3975 ±0.001	2880 ± 0.	0.1 ± 0.12
QRD	0.3215±0.06	1800 ± 1	0.16 ± 0.1	0.377 ±0.03	35 ± 2	0.1 ± 0.12
TOP	0.2707±0.0	338 ± 21	0.16 ± 0.1	0.4592 ±0.0	36 ± 2.3	0.1 ± 0.12
GRE	0.112 ± 0.0	2880 ± 0.	0.16 ± 0.1	0.5177 ±0.02	2880 ± 0.	0.1 ± 0.12

Figure 2: Quality and Running time

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.

Algorithm 3 Actor Critic Proximal Policy Optimization

Input: Action space A
Output: actor critic networks with parameters θ

```
while not to the max iterations do
    for actor = [1, ..., N] do
        Run policy π_{θ_old} in environment for T timesteps
        compute advantage estimates Â_1 ... Â_T
    end for
    Optimize surrogate L w.r.t. θ, with K epochs and mini-batch
    size M ≤ T · N
    θ_old ← θ
end while
```

5.1 Actor-Critic with Proximal Policy Optimization

In HARLM, the agent's policy takes the state representation as input and infers the optimal action for the next step. However, it is challenging to achieve good performance using a conventional policy network due to the potentially high variance in the returned rewards. To address this, we use the actor-critic method with entropy regularization. Entropy regularization fine-tunes the objective function to select more diverse tuples.

Typical policy-based reinforcement learning methods, akin to classic RL algorithms like REINFORCE [23], focus on optimizing the parameterized policy with respect to the expected long-term reward. Specifically, the objective of the policy network is to maximize $J(\theta)$, defined as follows:

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

where $\tau = (s1, a1, \ldots S_T, a_T)$, represents a trajectory leading to a selected approximation, i.e. $approx\_set = [a_1, ...a_T]$. The aim is to increase the probability of selecting the trajectory τ given a high reward $R(\tau)$. REINFORCE then uses gradient ascent to update the parameters θ in the direction of the gradient:

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

Motivation for actor-critic networks. Unfortunately, REINFORCE introduces high variability in the cumulative reward values ($\sum_{t=1}^{T} r_t$), resulting in instability and slow convergence during training [49]. To mitigate this, we use actor-critic networks. The actor network 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 [37].

We first introduce the forward pass of the actor and critic networks, and then explain how the networks are updated based on the reward and baseline.

The actor-critic network. The actor-critic network consists of two main components: the actor and the critic. The actor employs a policy function $\pi(a_t|s_t;\theta)$, parameterized by θ, to select actions, while the critic estimates action values using a value function $V^\pi(s_t;\theta_v)$, parameterized by θ_v. Since the value function cannot be computed exactly, it is estimated using a forward view and a mix of n-step returns to update both the policy and value function. The policy and value function are updated either after every t_{max} actions or when a terminal state is reached. Training stability is enhanced by using parallel actor-learners, accumulating updates for improved convergence.

Both actor and critic networks are implemented as neural networks (NN), with a large input layer matching the action space's size, followed by smaller fully-connected layers. These layers capture significant patterns between tuples in the database tables. In the critic network, a softmax layer outputs the policy distribution $\pi_\theta(s_t) = \text{Softmax}(\text{NN}(s_t))$, defining the action probabilities. For the critic network, a linear output provides the value function $V(s_t;\theta_V)$.

Our system trains the RL model with 32 actor and critic networks, asynchronously (as first presented at [40]). This parallelization enhances exploration by allowing actors to explore different parts of the environment independently. Different exploration policies are explicitly used in each actor-critic to maximize diversity. This approach stabilizes learning and offers practical benefits such as a reduced training time, which is linearly dependent on 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 [49].

Specifically, we replace the cumulative reward $\sum_{t=1}^{T} r_t$ by $A(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, 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:

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

where $A(s_t, a_t)$ can be estimated by $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$, called the *temporal-difference* (TD) error. From another perspective, since the baseline, i.e. the V value, is the expected long-term reward of a state, the TD error reflects the advantages and disadvantages of different actions from the 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 as small as possible. Hence, the TD error can also be used to update the critic, and the loss function can be written as $L_\varphi = (r_t + V^\pi(s_{t+1}) - V^\pi(s_t))^2$.

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 the adjusting the loss function to contain a proximal policy update, which is creating a *trust region* by clipping the *surrogate objective*. This results in the updates of the policy being more moderate, as the maximization of the surrogate objective without a constraint can lead to excessively large policy updates. The major difference is the use of gradient clipping, which constrains the policy updates to be within an ε range, thus preventing large updates that can cause instability.

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)$$

where $r_t(\theta)$ denotes the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. Maximizing the surrogate objective alone, $\hat{E}_t[r_t(\theta)\hat{A}_t]$, without a constraint leads to excessively 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*[20] 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 [51].

Overall training algorithm. Our approach is shown in Algorithm 3. REINFORCE first initializes the parameters (Line 1). In each iteration (Line 2-7), the system selects a batch of tuples from different tables using the learned policy (Line 3). Then it computes an estimate of the advantage function for each selected tuple (Line 4) to compute the loss and update the policy parameters (Line 6-7).

Finally, it outputs a well-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 and 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).

Gradual-Set-Learning (GSL) environment. Our system uses the specially crafted *gradual-set-learning* (GSL) environment, initialized with an empty set. An action involves adding several tuples from different tables, drawn from the action space created during the pre-processing phase. 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. An alternative environment for our use case is *drop-one* (DRP). In this environment, an RL agent concludes an episode when a terminal state is reached or when a predetermined horizon time limit is reached. The environment is initialized as a set of k tuples. Each iteration involves selecting a pair of actions: first, removing a tuple from the current state, and second, adding another tuple to the state. Alternatively, the agent may choose not to change the current set. After performing an action a_t and reaching a state S_{t+1} (comprising k tuples), the environment evaluates the score of the new state using the metric described in Section 3. The reward is calculated as the difference $score(S_{t+1}) - score(S_t)$. The episode concludes when the horizon is reached, with the default set to $100K - 500K$, adjustable by the user.

Discussion. The primary limitation of DRP is its vulnerability to becoming trapped in a non-optimal approximation set. Furthermore, the initialization phase is crucial and unstable, leading to performance variations. Despite exploring various initialization strategies, we could find none that completely alleviated these drawbacks. Several other environments were also examined, particularly those that combine GSL and DRP. However, none achieved results as good as GSL alone; additionally, they exhibited larger deviations, indicating less stability. For more details, see Section 6.

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 are given in Section 6.3, and ablation studies in Section 6.4

As our experiments show, the approximation sets chosen by HARLM are consistently better than alternative approaches for both non-aggregate (see Figure 2) and aggregate (see Figure 9) queries.

Environment	Agent	IMDB		MAS	
		Score	Total Time (min)	Score	Total Time (min)
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

Figure 3: Reinforcement Learning Ablation Study

Baseline	Memory Size (k)				Frame size (F)			
	1000	5000	10000	15000	25	50	75	100
HARLM	**0.69±0.0425**	**0.747 ±0.014**	**0.795 ± 0.025**	**0.864 ± 0.032**	**0.73 ± 0.325**	**0.69±0.043**	**0.631 ± 0.013**	**0.591 ± 0.024**
VAE	0.024 ± 0.0	0.024 ± 0.0	0.024 ± 0.001	0.036 ± 0.001	0.024 ± 0.0	0.024 ± 0.0	0.012 ± 0.001	0.012 ± 0.001
CACH	0.152 ± 0.073	0.20 ± 0.03	0.239 ± 0.045	0.254 ± 0.08	0.189 ± 0.01	0.172 ± 0.073	0.139 ± 0.062	0.1 ± 0.02
RAN	0.246 ± 0.032	0.435 ± 0.034	0.469 ± 0.031	0.501 ± 0.032	0.301 ± 0.036	0.246 ± 0.032	0.15 ± 0.041	0.11 ± 0.024
QUIK	0.296 ± 0.036	0.39 ± 0.021	0.47 ± 0.022	0.525 ± 0.051	0.315 ± 0.041	0.296 ± 0.036	0.175 ± 0.042	0.145 ± 0.031
VERD	0.387 ± 0.034	0.46 ± 0.082	0.54 ± 0.021	0.57 ± 0.017	0.34 ± 0.034	0.387 ± 0.034	0.26 ± 0.026	0.22 ± 0.051
SKY	0.340 ± 0.001	0.481 ± 0.001	0.551 ± 0.001	0.594 ± 0.0	0.446 ± 0.001	0.340 ± 0.001	0.247 ± 0.001	0.202 ± 0.001
BRT	0.347 ± 0.004	0.45 ± 0.003	0.542 ± 0.07	0.566 ± 0.003	0.42 ± 0.003	0.347 ± 0.004	0.25 ± 0.014	0.22 ± 0.009
QRD	0.350 ± 0.045	0.316 ± 0.065	0.432 ± 0.02	0.512 ± 0.05	0.368 ± 0.044	0.350 ± 0.045	0.303 ± 0.041	0.28 ± 0.061
TOP	0.364 ± 0.001	0.406 ± 0.001	0.482 ± 0.001	0.526 ± 0.001	0.401 ± 0.018	0.364 ± 0.001	0.257 ± 0.002	0.231 ± 0.001
GRE	0.29 ± 0.01	0.30 ± 0.02	0.33 ± 0.015	0.38 ± 0.021	0.29 ± 0.015	0.29 ± 0.01	0.17 ± 0.019	0.14 ± 0.03

Figure 4: 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% (Figure 4). 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 (Figure 2). 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 of our system over several datasets:

(1) *IMDB-JOB* - This dataset [27] contains both data and a query workload. The data is about movies and TV shows, and comes from IMDB. Its size is 34 million tuples.
(2) *MAS* - The MAS dataset [36] contains data about researchers and publications. Its size is 600K tuples. The query workload comes from [7].
(3) *FLIGHTS* - This dataset contains information about flight delays. The data is taken from [13] and queries are generated according to [26].

Baselines. To assess the effectiveness of ASQP-RL, we conduct comparisons with various baselines designed to address related tasks and potentially offer candidate solutions for the ASQP 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:

Naive baselines:

(1) *Random sampling (*RAN*)* - Randomly select rows from the large dataset.
(2) *Brute Force (*BRT*)* - An algorithm that exhaustively checks different combinations of k tuples to find the optimal solution. It evaluates all possible combinations to determine the best subset of tuples. To manage the computational complexity, a time constraint of 48 hours is imposed and parallel processing with 1000 processes is used to expedite the search process. We then return the best approximation set found during this process.
(3) *Greedy sampling (*GRE*)* - In each iteration, take the row that achieves the largest marginal gain with respect to the metric, eliminate this row, and repeat. The running time is limited to 48 hours and 500 processes are used to expedite the process.
(4) *Top queried tuples (*TOP*)* - Choose a random subset from each query answer. Choose queries that appear in the most queries first, until reaching k tuples.

Baselines from database domains:

(5) *Caching (*CACH*)* - Simulates a database's cache [3] by preserving tuples from the last executed query. This involves managing limited memory to store frequently accessed data, evicting the least recently used (LRU) pages to accommodate new ones.[†]
(6) *Query result diversification (*QRD*)* - An algorithm based on [31]. It follows an iterative approach where it selects the medoids of clusters and then re-assigns the data points to their nearest medoids.
(7) *Skyline (*SKY*)* - A technique for summarizing a dataset presented in [42]. While a skyline is typically used with numerical values, we extended it to handle categorical columns by comparing two values based on their frequency.
(8) Verdict *(*VERD*)* - A method for solving AQP proposed in the paper [44], that relies on sampling, rewriting queries and adjusting the answers returned from samples.
(9) QuickR *(*QUIK*)* - Another AQP method, proposed in the paper [24]. QuickR relies on sampling and table statistics to keep a catalog of plans and samples and an algorithm for choosing the right samples at the right time.

Generative model baseline:

(8) *Generative Model (*VAE*)* - We utilized a Variational Autoencoder (VAE) from [50] as a representative of state-of-the-art generative model for approximate query processing for *aggregate queries*. The VAE is used to generate fictitious tuples, and queries are executed on the generated data.

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 (for example, the Pandas python library uses a 20 row limit) but still small enough to be informative for a human. We experiment with different values for both parameters as will be shown later. Finally, by default the system executes all the queries given during training. Validation of these values is given in Section 6.4.

Environment. HARLM is implemented in Python 3.9.13 and is an open source library [6]. 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 [39] for Reinforcement Learning, OpenAI's Gym [9] for the environment interface and Pytorch [45] 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 begin by showing a simple experiment that shows how much time it takes to query the database directly, in order to highlight the need for solutions that address non-aggregate as well as aggregate queries. We use several versions of the *IMDB* dataset, each one larger then the other. We then query the databases with the workload in different orders and average out the time taken for each query, accumulating the results as the number of executed queries grow. As shown in Figure 6, even using the relatively small size of 1GB, on average we already pass the 5 hour mark after just seven queries. This justifies what was mentioned in Section 1, that lengthy exploratory queries are not uncommon, and lie beyond the scope of the typical query workload for which the database has been optimized.

6.2 Overall Evaluation

We compared our system's performance to that of the baselines, focusing on both quality and running time across various databases. To do this, we partition the workload Q into a training set (Q_{train}) and a test set (Q_{test}). For all baselines, the setup utilizes Q_{train} (if necessary), and the evaluation occurs over Q_{test} using the quality metric introduced in Section 3 (Equation 1) for SPJ queries and *relative error* for aggregate queries ([44],[50],[19]). We perform multiple train-test partitions, presenting both averaged results and variance. The results align with the default system parameters, and any deviations are investigated through an ablation study.

Non-Aggreagte Queries Quality Assessment. A comparative evaluation of our system's quality is presented in Figure 2. All baselines were limited to 48 hours; since the greedy (GRE) and brute-force (BRT) baselines failed to complete within this time, even when executed in a multi-process environment, we present the best results achieved within the imposed time constraint.

As evident in Figure 2, our method outperforms the baselines, even with an aggressive memory constraint of $k = 1000$ which is just a fraction of the full data (less then 0.0001% for *IMDB* and 0.001% for *MAS*). In addition, due to the different characteristics of each dataset some of the other baselines perform better on one while others perform better and the other.

This variance can be ascribed to the size and intricacy of the dataset and workload. "Naive" algorithms such as brute-force (BRT) and top-k (TOP) exhibited lower performance when used with larger

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

and more intricate workloads. Interestingly, the VAE algorithm obtained a subpar score due to its inability to generate tuples that aligned with query filters, and its failure to generalize over the workload. This accentuates the challenges in learning approximation sets for selection queries. Additionally, these results improve as the memory constraint increases while achieving a significantly improved running time (to be discussed later). Additional quality results for non-default parameters are discussed in Section 6.4. We also present the scores achieved by HARLM-light, an instantiation of HARLM with a reduced training set (25% of the original size) and a high learning rate of 0.1. Note that HARLM-light obtains a slightly lower score than HARLM.

Aggreagte Queries Quality Assessment. We compared our solution to recent leading approaches: gAQP [50], DeepDB [19] and DBEst++ [32]. 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 [13]. These queries were divided into train-test sets for evaluation. For gAQP, we recreated the experiments on the *FLIGHTS* and the *TPC-H* [1] datasets mentioned in [50], 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 [50] and [19]). 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.

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 in Figure 2, focusing on both the *set-up* ($setup$) and *query execution* ($QueryAvg$) time. The former represents the time (in minutes) the method requires to create the approximation set, while the latter signifies the time it takes to answer a set 10 queries from the workload. These measurements are averaged across 20 different query sets. 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. Divergence also exists in some baselines, notably generative approaches like the VAE, requiring additional time to provide an answer once a query is submitted by the user (approximately five minutes, compared to seconds for other baselines). Also note that HARLM-light (discussed earlier), 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, attaining 0.75 precision and 0.85 recall.

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 to the average of 70% shown in Figure 2. 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 handle scenarios where the query workload is not initially provided, we evaluate the system's performance using the *FLIGHTS* dataset. The system starts by generating several queries while simultaneously prompting the user to contribute a subset of queries for refining the generated ones. After incorporating the user-provided queries into our system and constructing an approximation set, we assess the quality of the approximated answers for the user's queries. As the user submits additional queries, the system refines its model through fine-tuning, leading to a performance increase. In this experiment, the user adds five queries at each step, helping the system generate queries more aligned with the user's interests and thereby fine-tuning the model on increasingly relevant queries. Figure 9 shows results exhibit a substantial improvement in answer quality starting at 70% for 10 queries reaching all the way to 90% for 50 queries, surpassing alternative methods. Our approach is compared against RAN and QRD, which are both capable of running without a workload. While RAN considers neither the data nor the workload, QRD uses inherent data patterns. However, QRD has lower scores than our method, not even reaching 70%.

Effect of Fine-Tuning. As discussed in Section 4.3, our system performs fine-tuning when it captures *interest drift* in user queries, which is done using the estimator. We start by dividing our workload

into three different subtasks using a clustering algorithm on the embedded version of queries, ensuring that the addition of new queries induces an *interest drift*. We then select a test sample from each cluster for querying the system. Initially, we train the system on the first cluster and query it with the corresponding workload. We gradually introduce parts of the test set matching the training workload the system hasn't seen. As anticipated, the estimator flags these queries as unanswerable, prompting fine-tuning by providing the training set of the second cluster. This process is repeated, incorporating the remaining training set. The results demonstrate a rapid enhancement in the quality of the created approximation set when fine-tuning is introduced, with results rising from 60% to 70% after finetuning is preformed (see figure 12)

Diversity. We also conducted experiments to assess the diversity in the approximated answers provided by our system versus those from other baselines. We initially considered adding a diversity factor to the overall score function, but eventually omitted it due to the impact on performance. We measured result diversity using a standard metric based on pairwise *Jaccard distance* among query answers. A comprehensive evaluation was conducted using the *IMDB* dataset, drawing inspiration from various diversity metrics in prior research (e.g., [5, 12, 48, 54]). For an average query in the dataset (when executed with LIMIT 100), the diversity in the answers retrieved from the database reached 58% on average. In contrast, our solution achieved an average diversity of 52%, which was at least 14% higher than any other baseline except for RAN. Notably, our approach competes closely with RAN, which randomly selects tuples from the DB but significantly lags behind in approximation's quality.

6.3 User Study

To subjectively verify our system's effectiveness, we followed the approach of prior AQP studies ([15, 38, 46]) 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 ([36]) 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 [27] (available at [6]). There, participants were instructed to formulate their own queries and document insights gained, akin to the methodology outlined in [4]. Participants were divided into two groups: one group 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 [6]. As shown in Figure 5 (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* [18]. The findings demonstrate that HARLM's users experienced reduced time pressure and frustration, and perceived greater success (Figure 5, right columns).

6.4 Ablation Studies and Parameter Optimization

Effect of Memory Size Constraint (k) on Quality. We studied how the performance of all baselines is influenced by changes in the memory size (k). In this experiment, we test for k in the range $[10^3, 5 \cdot 10^3, 10 \cdot 10^3, 15 \cdot 10^3]$. As expected, all tested approaches showed improvement as the size of the approximation set increases. The results in Figure 4 demonstrate that our method consistently outperforms other baselines. It achieves an average score of 80% at $k = 15 \cdot 10^3$, which is twice the score of GRE and 20% higher than SKY or QRD.

Effect of Frame Size Constraint (F) on Quality. Similar to the previous experiment, we tested the *frame size* (F) in the range $[25, 50, 75, 100]$. Increasing the frame size while keeping the memory size fixed makes the problem more challenging, as more tuples are needed for each query, leading to an overall decrease in quality. Despite this, ASQP-RL consistently outperforms other baselines, as shown in Figure 4. It also has fairly stable performance compared to, for example, SKY, which starts at 40% and decreases to 20%. It is important to note that a reasonable value for F is less than 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 10a, 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 10b, we see the training time for the same set of experiments, with training time decreasing to approximately 30 minutes.

RL Hyper-parameter Tuning. We tested the influence of tuning the main hyper-parameters of our RL algorithm, and present here the results in terms of quality. Figure 11 shows the effect of tuning the the learning rate (lr) in range $[5 \cdot 10^{-5}, 5 \cdot 10^{-4}, 5 \cdot 10^{-3}, 5 \cdot 10^{-2}]$, the entropy coefficient in range $[0, 0.001, 0.0015, 0.01, 0.015, 0.02]$, and the kl coefficient in range $[0.2, 0.3, 0.5, 0.7, 0.9]$. Notably, we conclude that the entropy coefficient has a crucial impact on the success of the algorithm and have set it to 0.001. The KL coefficient is also important although seems to have a lesser impact; we set it to 0.3. Finally, as is common in learning algorithms, a higher

learning rate can achieve higher convergence but at the expense of the model's stability.

Ablation Studies. Figure 3 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 architecture 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 our actor-critic architecture, along with clip loss, is crucial for achieving high system performance.

7 CONCLUSIONS AND FUTURE DIRECTIONS

We introduced the problem of approximating answers to non-aggregate queries and proposed a solution for approximating query workloads containing both aggregate and non-aggregates. 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 managing database updates, devising approximation sets with problem-specific constraints, and supporting approximate visualizations.

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

Figure 5: User study results

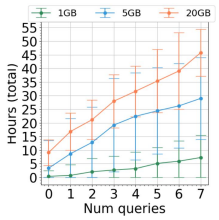


Figure 6: Query times

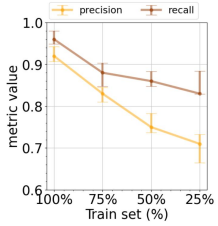


Figure 7: Estimator quality

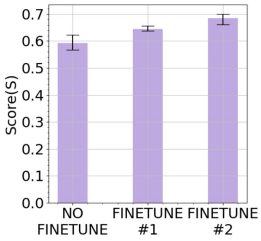


Figure 12: Finetuning the model

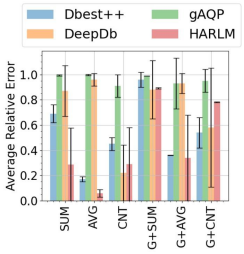


Figure 8a: Flights AQP

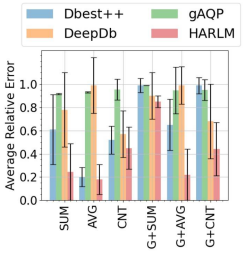


Figure 8b: TPC-H AQP

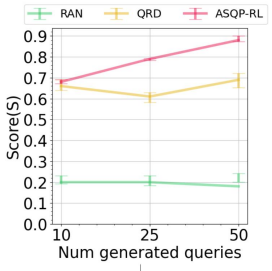
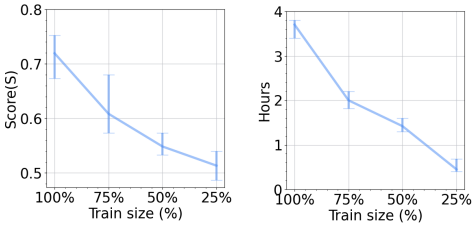


Figure 9: Unknown workload



(a) Quality (b) Time

Figure 10: Training set size

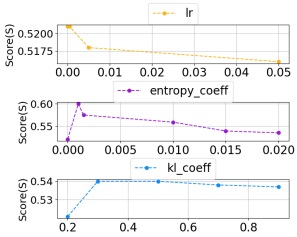


Figure 11: RL parameter tuning

REFERENCES

[1] [n.d.]. TPC-H dataset. https://www.tpc.org/tpch/. Accessed: 2024-04-16.

[2] 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.

[3] 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.

[4] 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.

[5] Yael Amsterdamer and Oded Goldreich. 2020. Diverse User Selection for Opinion Procurement. In *EDBT*.

[6] anonymous. 2023. *ASQP Github Repository*. https://anonymous.4open.science/r/SAQP-C90C/

[7] 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.

[8] 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

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

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

[11] 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.

[12] Marina Drosou and Evaggelia Pitoura. 2010. Search Result Diversification. *SIGMOD Rec.* 39, 1 (Sept. 2010), 41–47. https://doi.org/10.1145/1860702.1860709

[13] 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.

[14] 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.]).

[15] 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* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 1673–1682. https://doi.org/10.1145/2207676.2208294

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

[17] 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

[18] 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.

[19] 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).

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

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

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

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

[24] 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.

[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] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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

[32] 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)*.

[33] 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.

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

[35] 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).

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

[37] 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.

[38] 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, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 2904–2915. https://doi.org/10.1145/3025453.3025456

[39] 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.

[40] 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).

[41] 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.

[42] 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

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

[44] 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.

[45] 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).

[46] 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. *Proceedings of the VLDB Endowment* 10, 11 (1 Aug. 2017), 1262–1273. https://doi.org/10.14778/3137628.3137637 Publisher Copyright: © 2017 VLDB.; 43rd International Conference on Very Large Data Bases, VLDB 2017 ; Conference date: 28-08-2017 Through 01-09-2017.

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

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

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

[50] 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.

[51] Yingjie Tian and Yuqi Zhang. 2022. A comprehensive survey on regularization strategies in machine learning. *Information Fusion* 80 (2022), 146–166.
[52] 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.
[53] 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.
[54] 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).