Optimizing Source Selection for Tuple-Value Discovery

Ahmad Fares
CNRS, Univ. Grenoble Alpes
Saint Martin D'Hères, France
ahmad.fares@univ-grenoble-alpes.fr

Georgia Troullinou
CNRS, Univ. Grenoble Alpes
Saint Martin D'Hères, France
georgia.troullinou@univ-grenoble-alpes.fr

Silviu Maniu
CNRS, Univ. Grenoble Alpes
Saint Martin D'Hères, France
silviu.maniu@univ-grenoble-alpes.fr

Sihem Amer-Yahia
CNRS, Univ. Grenoble Alpes
Saint Martin D'Hères, France
sihem.amer-yahia@univ-grenoble-alpes.fr

ABSTRACT

In dataset discovery applications, users are interested in finding tuples in source tables containing attribute values of interest. We formulate TupleValDisc, a multi-objective tuple-value discovery problem that admits a set of source tables and a user request in the form of attribute values, and seeks to build a target table that *maximizes coverage* of requested attribute values, *minimizes penalty* incurred by unrequested values, and *minimizes the number of sources* used to build the target table. The order in which source tables are considered to address the user request calls for a sequential decision-making solution. We hence formulate a Markov Decision Process and propose a Reinforcement Learning algorithm to solve our problem. Our experiments corroborate the need for RL to solve the tuple-value discovery problem we introduced as a building block for more complex dataset discovery tasks.

VLDB Workshop Reference Format:
Ahmad Fares, Georgia Troullinou, Silviu Maniu, and Sihem Amer-Yahia. Optimizing Source Selection for Tuple-Value Discovery. VLDB 2025 Workshop: Tabular Data Analysis (TaDA).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts are available at https://github.com/AhmadFares/Source-Selection-for-Tuple-Value-Discovery

1 INTRODUCTION

Dataset discovery is the process of identifying and collating datasets. Its first purpose is to create a new, potentially virtual dataset. This may, for example, be done directly through a search, by navigating from related datasets, or by browsing the datasets with a specific annotation [12, 15, 21]. A fundamental building block of dataset discovery is *tuple-value discovery* where the purpose is to verify if attribute values of interest can be retrieved from source tables. The main challenge of tuple-value discovery is to identify source tables containing attribute values of interest and determine the order in which to scan those tables, without incurring too much noise, that is, without overwhelming the user with extra data that is not of interest to them. To achieve that, we propose an optimization problem formulation and a Reinforcement Learning (RL) solution.

Motivating example. Figure 1 shows a scenario where an instructor is searching for math-related questions to build a quiz. The sources S_1, S_2, and S_3 represent tables created by different instructors. Each row is a question characterized by a combination of attributes such as *keyword_name*, *topic_name*, and *subtopic_name*, with values like v_{11}, v_{21}, representing specific math concepts covered by the question (e.g., "Mean", "Linear Algebra", or "Vector Spaces").

A *User Request* (UR) specifies a set of values of interest across multiple attributes, for example:

$$\text{UR} = \{keyword\_name : \{v_{11}, v_{12}\},\ topic\_name : \{v_{21}\}\}.$$

That is, the user is looking to retrieve a set of questions that collectively cover these requested values, such as those tagged with v_{11} or v_{12} in *keyword_name*, or v_{21} in *topic_name*.

Consider the example source tables, S_1, S_2, and S_3, and user requests UR_0, UR_1, and UR_2, in Figure 1. For instance, UR_1 seeks attributes a_1 and a_2 with values $\{v_{11}, v_{12}\}$ and $\{v_{21}, v_{23}\}$ respectively. We assume that source tables are scanned row by row. When an attribute value in the user request is found in a table, we say that it has been *covered*. In general, given a user request, not all attribute values can be covered. That is for instance the case of UR_0 in which value v_{25} for attribute a_2 does not appear in any source table. However, for UR_1, both S_1 and S_2 contain requested attributes and values. The question in this case, is to determine which of these two source tables is better to use for UR_1. S_1 is better as it would only require reading tuples 1, 2, and 3 to generate target table T_1 that satisfies UR_1. Source S_2 would require scanning all its 4 tuples and retrieving unrequested attribute values: v_{20}, v_{22}, v_{13}, and v_{15}. Hence, using S_1 is better to satisfy UR_1, resulting in table T_1. While UR_1 can be satisfied using a single source, UR_2 would require scanning both S_2 and S_3 since none of the two tables contains all requested attributes and values. In this case, the challenge is to determine in which order to scan these tables to incur minimal overhead, both in terms of the number of tuples scanned and also in terms of reading data that is of no interest to the user. In this case, scanning S_3 then S_2 is preferred as it only requires reading 5 tuples (all of S_3 and then the first tuple in S_2) as opposed to 8 tuples (all of S_2 and then all of S_2). This would result in target table T_2 that best covers UR_2 and contains the fewest number of unrequested values.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

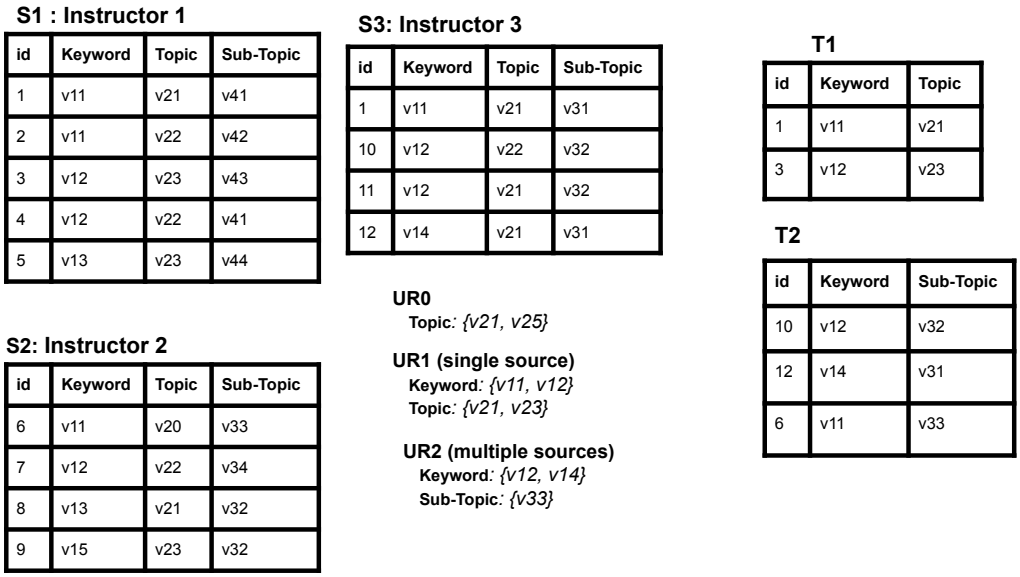


Figure 1: Source tables, user requests, and target tables.

Challenges of tuple-value discovery. We assume that source tables are not known or accessible in advance. Instead, they must be discovered and accessed dynamically, for instance in environments where schemas and contents are unknown beforehand. While the full contents are not available upfront, partial statistics about sources, such as attribute distributions, may be accessible.

Our goal is to build a target table given a user request. This raises multiple questions. The first challenge is to identify which subset of sources contains all the requested attributes and values. We refer to this dimension as *coverage*. The second challenge is to build a target table that satisfies the user request and contains as little unrequested data as possible for the given user request. We refer to this dimension as *penalty*. The third challenge is the number of source tables involved in satisfying a user request. Naturally, we would like to minimize that number. Finally, as shown in our motivating example, the order in which source tables are scanned incurs different coverage and penalties.

Contributions. Our first contribution is to formulate TupleValDisc, a multi-objective tuple-value discovery problem that admits a set of source tables and a user request, and generates a table that best covers input attribute values and reduces redundancy, formulated as minimizing penalty. The order in which source tables are considered to address the user request calls for a sequential decision-making solution. We hence formulate a Markov Decision Process and propose an RL algorithm that adapts a Deep Q-Network (DQN) implementation to solve our problem.

Our findings, using a real-world math learning dataset, show that using RL formulations for source selection in our problem case helps finding the best sources compared to offline baselines needing to traverse all sources. Moreover, we show that training of agents transfers between different distributions of sources and that RL agents still use much fewer sources than offline variants.

2 RELATED WORK

Dataset discovery involves locating relevant data based on specified user requirements. This can be achieved through direct search, by navigating linked datasets, or by browsing datasets based on specific annotations. Since requirements can be expressed in various ways, the approaches to dataset search tend to be diverse. Our work is related to two main areas: dataset discovery and the application of Reinforcement Learning (RL) for both multi-objective optimization and cost-based query optimization.

2.1 Dataset Discovery

Dataset discovery is a very active research area that covers dataset search, data navigation, data annotation, and schema inference [21]. Specifically, dataset search aims to retrieve datasets that, individually or in combination, fulfill a user-defined requirement. Once a relevant dataset is identified, it is often necessary to explore related datasets that can enrich the information or provide additional context through dataset navigation processes [19, 22, 36]. Data annotation systems [6, 10, 18] involve linking either specific data instances or general data descriptions to terms from a vocabulary or concepts from an ontology, helping to clarify the meaning and relationships within the data. In parallel, schema inference [13, 29] supports integration by identifying structural similarities across datasets and generating a unified schema.

In dataset search, requests may take the form of keywords, sample tables, or natural language queries, each requiring different approaches. This has led to the development of techniques that match these varied inputs to datasets in repositories by identifying syntactic or semantic similarities. The most extensively studied cases involve keyword-based and dataset-based queries. In a keyword-based search [2, 35], users submit a set of keywords, and the system

returns a ranked list of relevant datasets, including semantic matching techniques [4, 23]. Dataset-based queries approaches, where the search is conducted over the headers and/or content of the dataset, take as input a dataset and return related datasets, using table union search [17] or join-correlation search [1, 25] There are also table augmentation techniques that, given a dataset, search for others to enrich it with additional attributes or rows [31, 34].

Another line of work focuses on reconstructing a target table from existing ones. Gen-T [7] addresses this table reclamation problem, where the objective is to reconstruct a user-specified target table by composing data from multiple tables in a data lake. Table reclamation leverages semantic table embeddings and clustering to identify and assemble relevant tables, operating at both the schema and content levels. We take here a different approach, one in which the target table is not known in advance, only its schema and desired attribute values.

The recent integration of large language models (LLMs) into dataset discovery is motivated by the need to address some issues in the traditional approaches [12]. LLMs have enabled the handling of complex and ambiguous user requests, offering more semantically meaningful dataset retrieval. In [9], the authors propose an automated annotation framework using LLMs to efficiently generate labeled datasets for dataset search tasks by combining table compression and result aggregation techniques. The approach in [27] introduces a pipeline that leverages LLMs to generate synthetic query-description pairs for fine-tuning dense retrieval models in dataset search. Solo [30] is a self-supervised table discovery system that automatically constructs training datasets from table repositories to train models that retrieve tables containing the answers to natural language questions. The authors in [3] use GPT-3.5 Turbo to decompose natural language queries into sub-queries, which may correspond to different tables and columns, enabling retrieval and joining of multiple tables when the join plan is not explicit in the original query. This line of work has also extended to the use of small language models (SLMs), which offer lightweight and effective alternatives for retrieval tasks. ELEET [15] is a dataset discovery system designed to retrieve relevant tables from a large corpus based on a user request, leveraging a fine-tuned SLM for efficient similarity-based matching. Recent table union search methods [8, 11] utilize pre-trained language models to better capture and represent column semantics. Joinable table search finds tables that can be joined with a query table to enrich it with more attributes. Specifically, the authors of [5] propose a deep learning framework that tackles the challenge of joining semantically related columns by fine-tuning to learn value transformation rules that enable effective alignment for joins. [16] proposes an LLM-based framework for causal dataset discovery, aiming to identify columns with potential causal relationships across correlated tables. The work in [20] proposes a data-driven method for discovering semantic domains by identifying sets of terms across a collection of tables through the analysis of value co-occurrence information across dataset columns.

Our work aligns with the notion of *"data on demand"* in dataset discovery, retrieving only the data needed by the user at query time. We introduce the atomic notion of *"tuple-value discovery"* and unlike existing work, our approach models *user request satisfiability* by adding notions of coverage and penalty, aiming to retrieve only the data that best satisfies the user request while minimizing redundant information.

2.2 RL for optimization

Model-free Reinforcement Learning (RL) [28] is a popular approach for solving sequential optimization problems whereas an agent acts on its environment and receives a reward for each action it takes, transitions in different states, and optimizes the cumulative reward. In our work, we aim to optimize the satisfaction of a user request by covering as many requested values as possible with the least redundant data and with the fewest number of source tables. These goals are not correlated which results in a multi-objective optimization problem.

Multi-Objective Reinforcement Learning [32] provides a principled framework for learning policies that provide a trade-off between multiple objectives. In [32], the authors apply scalarization to transform multiple objectives into a single scalar value via a weighted combination of the objectives. We adopt the same approach in our case. In [14], the authors explore the use of Deep RL to solve a multi-objective problem. The optimal solutions are obtained through forward propagation of the trained network. The network model is trained through the trial and error process of DRL and is used as a black-box heuristic or a meta-algorithm with strong learned heuristics. The use of DRL to optimize our objectives is left for future work.

Another related line of work related to ours is the use of RL for query optimization. For instance, HybridQO [33] is a hybrid optimizer that combines both cost models and learning techniques to improve join order in SQL. Our work applies a similar decision-making process where the order in which source tables are chosen bears similarity to join ordering. The applicability of this multi-objective framework in our case could be explored in the future.

3 PROBLEM AND SOLUTION

We are given a set of input source tables $\mathbf{S}$ consisting of attributes in a set $\mathbf{A}$, and a user request UR in the form $\{(a, \{v\})\}$ containing a set of pairs depicting the desired values $\{v\}$ for attribute $a \in \mathbf{A}$, where a is present in all sources $S \in \mathbf{S}$. Unlike a SQL selection query, a user request is not a filter over tuples. It specifies values to be collectively covered across multiple rows, not necessarily matched within a single tuple. Figure 1 shows an example of 3 source tables and 3 user requests.

3.1 Definitions

We define the notions of coverage and penalty, in relation to a given user request UR and a target table T.

We denote by $\texttt{values}(a, T)$ the set of unique values that appear in attribute a of table T.

Definition 3.1. **Per-Attribute Coverage (AttrCov).** The fraction of values in the user request UR that is contained in T for a specific attribute a:

$$\text{AttrCov}(UR, T, a) = \frac{|\text{values}(a, T) \cap \text{values}(a, UR)|}{|\text{values}(a, UR)|} \tag{1}$$

Definition 3.2. **Table Coverage** is computed as the average across per-attribute coverage in T:

$$\text{Coverage}(UR, T) = \frac{1}{|A|} \sum_{a \in A} \text{AttrCov}(UR, T, a) \tag{2}$$

To capture irrelevant data, we have to define the notion of *penalty* on the attribute values that do not appear in the user request UR but are contained in T.

Definition 3.3. **Per-Attribute Penalty (AttrPen).** The fraction of retrieved values in T for attribute a that were not part of the user request UR:

$$\text{AttrPen}(UR, T, a) = \frac{|\text{values}(a, T) - \text{values}(a, UR)|}{|\text{values}(a, T)|} \tag{3}$$

Definition 3.4. **Table Penalty** is computed as the average per-attribute penalty across all attributes:

$$\text{Penalty}(UR, T) = \frac{1}{|A|} \sum_{a \in A} \text{AttrPen}(UR, T, a) \tag{4}$$

Our coverage measure corresponds to precision, while penalty is inversely related to recall, but evaluated at the value level instead of tuple level, in accordance with our UR semantics.

Other aggregations could be explored in the future. For instance, minimizing variance across attribute penalties could be used to define table penalty.

3.2 Problem Statement

We are now ready to formulate our problem that aims to optimize the three dimensions of our setting (coverage, penalty, and number of source tables):

Problem 1 (TupleValDisc). *Given a set* $\mathbf{S}$ *of source tables* $\{S_1, \ldots, S_M\}$ *and a user request UR, the objective is to retrieve tuples to build a table* T *whose schema is defined by all the attributes in UR, denoted as* $proj_A(\mathbf{A})$*, by adding only tuples from* $S = \cup_i S_i$*, such that Coverage*(UR, T) *is* maximized*, and Penalty*(UR, T) *and the number of sources used* $|S|$ *is* minimized.

3.3 Markov Decision Process Formulation

Our goal is to rely RL approach to solve TupleValDisc, where the agent acts on its environment and completes the smallest table T that addresses the user request UR by maximizing the cumulative reward. There are four main elements of an RL process: a policy, a reward signal, a value function, and, optionally, a model of the environment.

A *policy* is a mapping from perceived states (i.e., sources) of the environment to actions to be taken in those states. In some cases, the policy may be a simple function or lookup table, whereas in others it may involve a search process as is the case in solving TuplevalDisc. As the reward function defines the goal of an RL problem, the sole objective is to maximize the total reward it receives and the reward signal is the primary basis for altering the policy. While the reward signal conveys immediate goodness, a value function indicates long-term desirability. In our case, the agent maximizes the total reward when it starts from a random source table and gets closer, at each step, to the user request UR. The last element of RL is a model of the environment, enabling the system to predict its behavior. For instance, when presented with a state and an action, the model can anticipate the subsequent state and the associated reward. In contrast, in our work, our focus is on model-free approaches that explicitly rely on trial-and-error methods.

We model this using a Deterministic Discrete Markov Decision Process (MDP) defined by a triple $\{\mathcal{S}, \mathcal{A}, \mathcal{R}\}$ composed of: the state space $\mathcal{S}$ of the environment; the action space $\mathcal{A}$ from which the agent selects an action at each step; and a reward function $\mathcal{R}$ that computes the reward of an action act_i from state st_i to st'_i, $R_i = r(\text{st}_i, \text{act}_i, \text{st}'_i)$. In this context, the defintions of *policy* and *optimal policy* are as follows:

Definition 3.5. **Policy π.** A *policy* $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ of an RL agent maps the probability of taking action $\text{act} \in \mathcal{A}$ in state $\text{st} \in \mathcal{S}$, that is, $\pi(\text{st}, \text{act}) = Pr(\text{act}_t = \text{act} | \text{st}_t = \text{st})$.

Definition 3.6. **Optimal policy π^*.** A policy π^* is optimal iff its expected cumulative reward is greater than or equal to the expected cumulative reward of all other policies π. The optimal policy has an associated optimal state-value function, as well as an optimal action-value function, or optimal Q-function, as defined below:

$$Q^*(\text{st}, \text{act}) \leftarrow max_\pi Q_\pi(\text{st}, \text{act})$$

In other words, Q^* gives the largest expected return achievable by any policy for each possible state-action pair.

We now detail the MDP ingredients for our setting.

States and actions. We define the environment of our source exploration agent as containing the set of source tables $\mathbf{S}$ and algorithms (presented below) that select tuples from a given set of sources S . The (discrete) action space is of size $|\mathbf{S}|$, where each action corresponds to a source S_i in the set of all available sources.

When an action is chosen, the corresponding source S_i is added to the candidate source set S the environment applies the tuple selection algorithm, `Complete_from_Source` algorithm, listed in Algorithm 1, to extract useful tuples from the source selected at step i, S_i. The selected rows are merged into $T(i)$, via the `Optimize_Selection` algorithm (as described below). The state at each step consists of the current bitmap of chosen source set S, source-level statistics, and the partially constructed table $T(i)$.

The `Complete_from_Source` algorithm is composed of three steps. The first step, `Coverage_Guided_Tuple_Selection`, is provided in Algorithm 2. It optimizes coverage by adding into T the tuples that ideally cover the entirety of the attribute values in the user request UR; if this is not possible, the algorithm stops when a coverage threshold θ is reached. Once the coverage optimization has been performed, the next step is refining T via `Penalty_Optimization` in Algorithm 3. This is performed by replacing a tuple in T with a tuple in S only if the incurred penalty is improved, i.e., reduced. The final step removes redundant tuples in T via `Optimize_Selection` depicted in Algorithm 4. This algorithm iteratively attempts to remove tuples from T and tests whether the coverage does not change; if this is the case, the tuple is removed (note that the penalty cannot increase in this case). The penalty optimization algorithm is run separately for two main reasons. The first is that, even if it could have been included in Algorithms 2 and 3, running it separately ensures that the optimization is only done once for each source.

Secondly, the algorithm is used as a final optimization step when results are joined to previous step in the RL sequence.

The computational complexities of the algorithms are: $O(S)$ for `Complete_from_Source`, $O(TS)$ for `Penalty_Optimization`, and $O(T^2)$ for `Optimize_Selection`. Considering that T is usually small (it can contain at most $|UR|$ tuples), our algorithms are very efficient in practice.

Algorithm 1: `Complete_from_Source`

Input: Source Table S, User Request UR, Desired Coverage Threshold θ
Output: Completed Table T

$T \leftarrow \emptyset$, $i \leftarrow 0$;
$(T, i) \leftarrow$ `Coverage_Guided_Tuple_Selection`(S, UR, θ) ;
$T \leftarrow$ `Penalty_Optimization`(S, T, UR, i, θ) ;
$T \leftarrow$ `Optimize_Selection` (T, UR) ;
return T ;

Algorithm 2: `Coverage_Guided_Tuple_Selection`

Input: Source Table S, User Request UR, Coverage Threshold θ
Output: Partial Table T, Index i

$T \leftarrow \emptyset$, curr_coverage $\leftarrow 0$, curr_penalty $\leftarrow 0$;
for $i \leftarrow 1$ **to** $|S|$ **do**
  $T_{\text{curr}} \leftarrow T \cup \{t_i\}$;
  if *Coverage*$(UR, T_{curr}) \leq \theta$ ***and*** *Coverage*$(UR, T_{curr}) >$ *curr_coverage* **then**
    $T \leftarrow T_{\text{curr}}$;
    curr_coverage $\leftarrow$ Coverage(UR, T) ;
    curr_penalty $\leftarrow$ Penalty(UR, T) ;
  if *curr_coverage* $\geq \theta$ **then**
    break ;
return (T, i) ;

When an agent visits a state st, it tries to navigate into a better state st′ by applying an action that selects one source from the available set S. At each step t, once `Complete_from_Source` is run, the new state is then the set of the chosen sources $S = \cup_{i \in 1,\ldots,t} S_i$ plus the target table T containing the chosen tuples. No source S_i should be selected twice in our case as this would not change the state, both in terms of S and T. Hence, our setting can be considered as one containing non-repeating actions.

Reward design. As TupleValDisc is a multi-objective problem, in the general case our reward will be a function of the coverage, penalty, and the number of selected sources:

$$R(UR, T, S) = f\left(\text{Coverage}(UR, T); \text{Penalty}(UR, T); S\right).$$

Similarly to [32], we propose to define our reward using scalarization. We hence write our reward as a weighted linear combination of three objectives:

$$R(UR, T, S) = \alpha \cdot \text{Coverage}(UR, T) - \beta \cdot \text{Penalty}(UR, T) - (1 - \alpha - \beta)\frac{|S|}{|\mathbf{S}|},$$

Algorithm 3: `Penalty_Optimization`

Input: Source Table S, Current Table T, User Request UR, Reached Index i, Coverage Threshold θ
Output: Updated Table T

curr_penalty $\leftarrow$ Penalty(UR, T) ;
if *curr_penalty* $= 0$ **then**
  return T ; `// Stop if penalty is zero`
for i **to** $|S|$ **do**
  foreach *row* $t_j \in T$ **do**
    $T_{\text{curr}} \leftarrow (T - \{t_j\}) \cup \{t_i\}$;
    if *Coverage*$(UR, T_{curr}) \geq \theta$ ***and*** *curr_penalty* $>$ *Penalty*(UR, T_{curr}) **then**
      $T \leftarrow T_{\text{curr}}$;
      curr_penalty $\leftarrow$ Penalty(UR, T) ;
      if *curr_penalty* $= 0$ **then**
        return T ; `// Stop if penalty is zero`
return T ;

Algorithm 4: `Optimize_Selection`

Input: Table T, User Request Table UR
Output: Optimized Table T

orig_coverage $\leftarrow$ Coverage(UR, T) ;
changed $\leftarrow$ `True` ;
while *changed* **do**
  changed $\leftarrow$ `False` ;
  foreach *row* $t_j \in T$ **do**
    $T_{\text{sub}} \leftarrow T - \{t_j\}$;
    if *Coverage*(UR, T_{sub}) = *orig_coverage* **then**
      $T \leftarrow T_{\text{sub}}$;
      changed $\leftarrow$ `True` ;
      break ; `// Restart loop after modification`
return T ;

where $\alpha, \beta \in [0, 1]$ control the relative importance of each component, with $0 < \alpha + \beta < 1$, and $\frac{|S|}{|\mathbf{S}|}$ represents the normalized steps in the episode. This formulation encourages the agent to find source tables that satisfy a UR with high coverage and low penalty, while also being efficient in the number of sources used.

To this reward signal, we add some optimizations. First, we add a STOP action to the action space, representing the decision of the agent to stop the episode when the reward is optimized. The reason for this choice was that we initially considered terminating an episode automatically once a threshold on coverage and/or penalty were reached. However, this approach would have imposed rigid limits on exploration and may have prevented the agent from discovering better coverage/penalty trade-offs. Note that the formulation of the reward encourages the agent to stop only when coverage is sufficiently high, and penalizes early stopping. We penalize with a low reward the cases when the first chosen action is

STOP and when the agent tries to select a source several times in the episode; we also stop early when coverage is 1 and penalty is 0, representing the ideal case.

3.4 Revisited Problem Formulation

We now reformulate Problem 1 (TupleValDisc) for our RL setting:

Problem 2 (RLTupleValDisc). *Given a set of source tables* S *and the task of incrementally building a target table* T *according to a user request* UR*, the objective is to find an optimal policy that maximizes its cumulative reward as follows:*

$$\pi^* = argmax_\pi R(UR, T, S).$$

Problem 2 refines Problem 1 to address settings where scanning all sources sequentially is impractical, and a dynamic selection process based on table statistics can be trained to reduce cost and improve efficiency.

3.5 Deep Q-Network Solution (DQN)

We use Deep Q-Network [28], a popular RL algorithm that extends Q-learning using deep neural networks to approximate the Q-value function. It aims to find an optimal policy for action selection in an MDP by estimating the expected cumulative reward for each $\langle state, action \rangle$ pair. Through an iterative process that balances exploration and exploitation, it updates Q-values based on observed rewards and future value estimates, enabling the agent to make increasingly informed decisions and maximize long-term rewards:

$$Q(\text{st}, \text{act}) \leftarrow Q(\text{st}, \text{act}) + \lambda \left[R + \gamma \max_{\text{act}' \in \mathcal{A}} Q(\text{st}', \text{act}') - Q(\text{st}, \text{act}) \right],$$

where $\lambda \in [0, 1]$ is the learning rate, which determines how many new experiences adjust old estimates, and $\gamma \in [0, 1]$ is the discount factor, controlling the importance of future rewards. We used the solution described in [26], PER (Prioritized Experience Replay Buffer), to define our replay buffer, which defines an effective way to acquire diverse experience during training.

4 EXPERIMENTS

The purpose of our experiments is to examine RL solution's effectiveness compared to three offline variants using the algorithms from Section 3.3 that iterate through the sources in random order.

4.1 Experimental Setup

Setup. The inference experiments were conducted on a Dell Precision 3561 running Debian, equipped with an 11th Gen Intel Core i7-11850H CPU, 31 GB of memory, and a 1 TB NVMe SSD. Code and data are available at: https://github.com/AhmadFares/Source-Selection-for-Tuple-Value-Discovery

Dataset. We use the MathE [1] dataset, a real-world dataset, available on an e-learning platform, that focuses on enhancing mathematical skills in higher education and supporting autonomous learning through 1,900 human-authored multiple-choice questions. The dataset contains 15 topics from which we selected 10 topics, e.g., Linear Algebra, Probabilities. The difficulty level of each MCQ has a numerical scale within [1,6], 1 being the easiest. The dataset contains 20,255 tuples reflecting student, question interactions, and 31 attributes. Notable attributes in this dataset for targeted knowledge search include *keyword-name*, *topic-name*, and *subtopic-name*, which enable users to request information related to students' performance on various mathematical concepts. However, highly specific and focused data may be valuable, especially when the request is described using as many attributes as possible.

To simulate realistic scenarios where users seek data with varying levels of detail or specificity, we consider two URs:

- **Deep UR:** a user request that contains a few attributes, each with multiple requested values.
- **Shallow UR:** a user request that involves a larger number of attributes, each with only a few associated values.

These URs enable us to compare how our methods perform under both focused and vague information needs.

Source Generation. To simulate diverse real-world data conditions and introduce more challenging sources for our approach, we generate synthetic source tables from our dataset using the following strategies:

- **Noise Injection.** We randomly remove 30% of the values requested in a *UR* from the original dataset/table and replace them with unrequested values as noise. Afterwards, the resulting table is split into distinct, equally sized sources by distributing its rows based on randomly chosen allocations. As a result, the sources are intentionally designed to make full coverage of the user request unachievable. We refer to the agent trained with this strategy as RLNoiseInjection.
- **Targeted Noise Injection.** For rows in which all attribute values satisfy the input user request *UR*, one value is randomly replaced with an unrequested value. The resulting corrupted table is then divided into distinct, equally sized sources according to randomly assigned splits. The agent trained using this strategy is called RLTgtNoiseInjection.
- **UR Values Injection.** First, we construct an optimal table based on the input user request (*UR*) by taking the Cartesian product of all values in *UR*, producing rows in which every value satisfies *UR*. This results in a table with zero penalty. The rows in this table are randomly injected into the initial source table. The resulting table is then divided into distinct sources of equal size by randomly selecting portions of rows. We denote the agent trained with this strategy RLValueInjection.

For the MathE dataset and for each of the above strategies, we split the tuples in the original table into 10 sources of comparable size. Handling source tables of different sizes is left to future work.

Measures. For training, we measure the evolution of rewards, number of steps per episode, and the choice of the STOP action. For inference, we report the order in which the sources are selected, we measure coverage and penalty of the target table, number of sources selected, as well as running time.

We also report the result of transfer learning from one set of sources to another, namely, we examine if an agent trained for a set of sources, performs well on using another set of sources for the same request.

[1] https://mathe.ipb.pt/

4.2 Offline Variants

To assess the effectiveness of our RL solution, we define three offline variants using the algorithms in Section 3.3 that loop through the sources in S in an arbitrary order:

- **One-Pass Coverage Variant (No Optimization).** This variant stops selecting rows as soon as the coverage threshold θ is reached. It does not perform any additional optimizations. This corresponds to the `Coverage_Guided_Tuple_Selection` algorithm and is denoted as OfflineCoverage.
- **Penalty Optimization.** After achieving coverage θ, this variant attempts to replace existing rows in the selected subset with better candidates to minimize the penalty while maintaining the required coverage. This corresponds to `Coverage_Guided_Tuple_Selection` followed by the `Penalty_Optimization` algorithm, and is denoted as OfflinePenaltyOpt.
- **Overall Selection Refinement.** In addition to penalty optimization, this variant includes an optimization step that further refines the target table by removing rows that do not contribute to coverage. This ensures a minimal and efficient selection, and corresponds to the entire `Complete_from_Source` algorithm. It is denoted as OfflineAll.

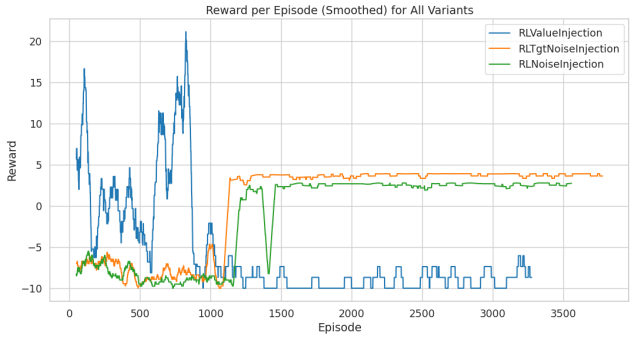
These offline variants serve as comparison to the RL agents which execute at each step `Complete_from_Source` algorithm only on the sources chosen up to that point.

4.3 Results

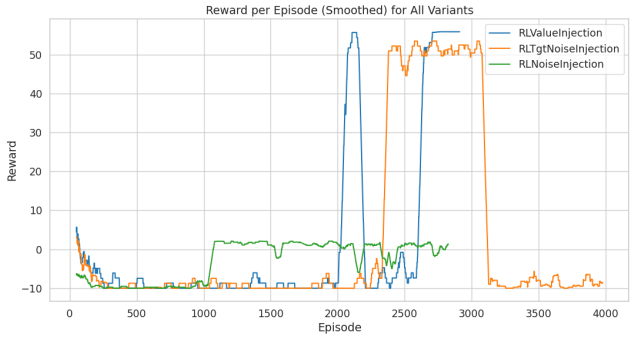
We set in our experiments $\alpha = 0.6$, $\beta = 0.3$; this is to have larger focus on coverage and penalty while accounting for the number of sources selected (set to 0.1). The values for each dimension are normalized to a value of 10. As described in Section 3.3, several penalties were introduced. A penalty of -10 is applied both when the agent selects the same source more than once (which also terminates the episode) and when it chooses STOP as its first action. Additionally, if the agent chooses STOP with full coverage of 1 and penalty of 0, it receives a large reward of 50 to indicate that it should not continue the optimization; however, such a scenario is rare in practice. We trained each agent for 7,000 timesteps using the Stable-Baselines3 DQN implementation, with the default learning rate (1×10^{-4}) and an exploration schedule that starts with fully random actions at the beginning of training, gradually shifting towards mostly greedy (best-known) actions as training progresses. The exploration rate is decreased over 40% of the training, encouraging the agent to explore initially and then exploit what it has learned.

Training. For training on the MathE dataset, we use a Deep *UR* with three attributes, each requesting four distinct values, and a Shallow UR consisting of 7 attributes, each having one or two values. The details of the *URs* are found in Appendix A. For each *UR*, we trained three agents each corresponding to the source generation strategies outlined in Section 4.1.

Our experiments show that RL agents trained on different source generation strategies learn to avoid repeated sources and to balance the trade-off between coverage, penalty, and the number of steps. However, the agent RLValueInjection trained on the Deep UR has difficulties learning optimal stopping. To address this, we propose a modified environment that removes the termination penalty for repeated source selection and instead ignores repeated sources in the reward, thus encouraging the agent to focus on maximizing reward without the added complexity of learning when to stop. Furthermore, we find that including the objective of source minimization in the reward is essential for efficient source selection policies. These findings guide our detailed analysis which follows.



(a) Deep UR



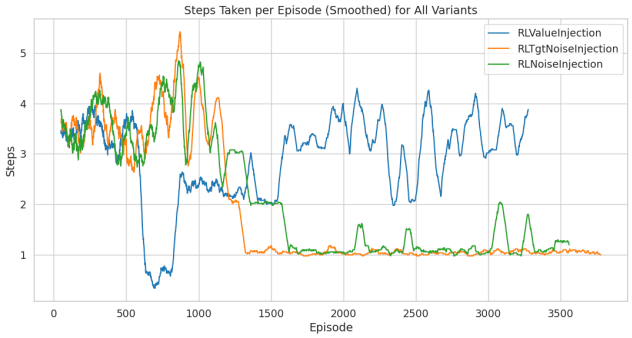
(b) Shallow UR

Figure 2: Reward evolution when training RL agents.

Figure 2a shows the evolution of the reward of agents trained on the Deep *UR* over the course of training episodes. As clearly seen in the figures, both RLTgtNoiseInjection and RLNoiseInjection maximize their reward after 1,300 episodes and converge on a reward ranging between 0 and 5, suggesting that agents learned to (i) avoid repeated sources and (ii) stop after a coverage-penalty trade-off is achieved. However, RLValueInjection starts by alternating between high and low rewards, then converges to a low reward, indicating that the agent kept selecting sources repeatedly, leading to a low reward, and never learned how to stop.

Figure 2b shows the reward evolution for agents trained on the Shallow *UR*. The agent RLTgtNoiseInjection achieves results similar to its counterpart trained on Deep *UR*, whereas other agents had different behavior. In particular, RLTgtNoiseInjection discovers that a coverage of 1 is achievable, which it successfully exploits between episodes 2300 and 3300. The drop in the end results from the agent attempting to further minimize the number of steps, which reached 0 as visible in Figure 3b, triggering a penalized reward. Furthermore, the difficulty in learning when to stop, previously

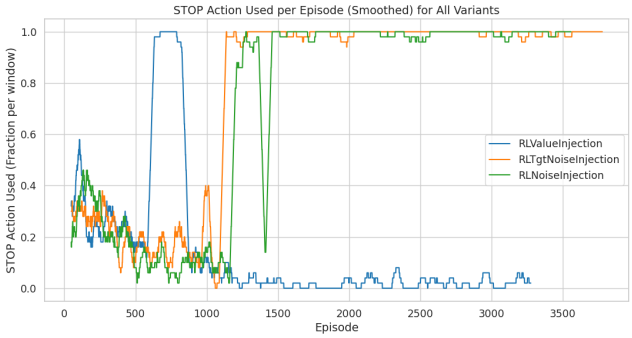
observed in the Deep *UR* setting, also affects RLValueInjection as shown in 4b.



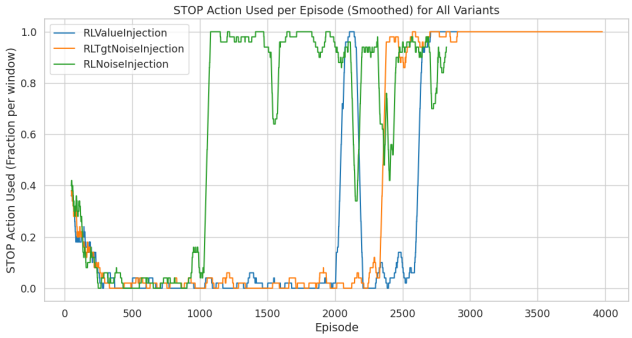
(a) Deep UR

(b) Shallow UR

Figure 3: Steps per episode for each RL agent.



(a) Deep UR



(b) Shallow UR

Figure 4: Number of STOP actions per episode for RL agents.

Figures 3a and 3b track the number of steps per episode. RLTgtNoiseInjection and RLNoiseInjection in both experiments learned that higher reward can be achieved in episodes having fewer steps. On the other hand, the graph of RLValueInjection trained on Deep *UR* shows more exploratory steps in later episodes, whereas in the Shallow *UR* setting, it behaves similarly to the other agents. These results are consistent with the results in Figure 4a and 4b, where we track the use of the STOP action in each episode. It can be seen that all agents consistently learned to use the STOP action, except RLValueInjection in the Deep *UR* case.

In order to address the limitation that the RLValueInjection agent fails to learn when to stop, we modified the environment so that selecting the same source twice no longer terminates the episode, and gives a reward of 0 (previously −10). This encourages the agent to focus on maximizing reward without the overhead of learning how to stop. This change requires increasing the number of episodes to 30,000 to accommodate potentially much longer episodes; whereas previously, the maximum steps per episode was the number of sources plus one. Now, episodes terminate only when the agent chooses STOP.

In Figure 5 (top) the results show an improvement in the behavior of the RLValueInjection agent, which converged to receiving the highest reward after approximately 5,500 episodes. The behavior of the two other agents is similar to that observed in the initial experiment. However, the decrease in reward may be due to increased exploration, leading to episodes with a large number of steps. Moreover, the difference in the number of episodes between the agents suggests that the RLValueInjection agent tends to perform shorter episodes compared to the other two agents, as confirmed by the steps count in Figure 5 (bottom).

Moreover, we conducted an experiment where the weight associated with the normalized steps, specifically $\frac{|S|}{|\mathbf{S}|}$, was set to zero by choosing $\alpha + \beta = 1$. This effectively eliminates any penalty related to the number of steps used per episode. All other parameters and experimental conditions were kept the same as in the initial setup. The results, shown in Figure 6, show that in this case the number of steps per episode does not decrease over time. This suggests that the absence of even a small weight on normalized step usage negatively impacts the agent's ability to learn efficient choice of sources. The results highlight the significance of including this weight to encourage shorter, more efficient source choice trajectories.

Inference. Once the training has been performed for each *UR*, we evaluate the resulting agents by using the trained policies on sources re-split using the three strategies detailed in Section 4.1. Hence, each agent is tested on all strategies but having different sources than the ones used for its training.

The results are summarized in Table 1. All RL agents consistently outperform offline methods in terms of runtime, except for instance

Table 1: Evaluation of offline baselines to online RL agents

Test UR + Source Split	Offline Coverage	Offline PenaltyOpt	Offline All	RL Noise Injection	RL Value Injection	RL TgtNoise Injection
Deep UR + Noise Injection	Coverage= 0.67 Penalty = 0.59 Time = 0.147 Steps = 10	Coverage= 0.67 Penalty = 0.59 Time = 0.143 Steps = 10	Coverage= 0.67 Penalty = 0.59 Time = 0.144 Steps = 10	Coverage= 0.67 Penalty = 0.59 **Time = 0.045** **Steps = 1**	Coverage= 0.67 Penalty = 0.59 **Time = 0.087** Steps = 3	Coverage= 0.67 Penalty = 0.59 **Time = 0.019** **Steps = 1**
Deep UR + UR values Injection	Coverage=1 Penalty= 0.494 Time = 0.011 Steps=1	Coverage=1 Penalty= 0 Time = 109.77 Steps=10	Coverage=1 Penalty= 0 Time = 110.83 Steps=10	Coverage=1 Penalty= 0.320 Time = 5.7 Steps=1	Coverage=1 Penalty= 0.066 Time = 14.75 Steps=3	Coverage=1 Penalty= 0.320 Time = 5.73 Steps=10
Deep UR + Targeted Noise Injection	Coverage=0.91 Penalty= 0.565 Time = 0.188 Steps=10	Coverage=0.91 Penalty= 0.565 Time = 0.186 Steps=10	Coverage=0.91 Penalty= 0.565 Time = 0.186 Steps=10	Coverage=0.91 Penalty= 0.565 Time =0.051 Steps=1	Coverage=0.91 Penalty= 0.565 Time = 0.147 Steps=3	Coverage=0.91 Penalty=0.565 Time = 0.03 Steps=1
Shallow UR + Noise Injection	Coverage= 0.071 Penalty = 0.857 Time = 0.169 Steps = 10	Coverage= 0.071 Penalty = 0.857 Time = 0.157 Steps = 10	Coverage= 0.071 Penalty = 0.857 Time = 0.156 Steps = 10	Coverage= 0.071 Penalty = 0.857 **Time = 0.077** Steps = 2	Coverage= 0.071 Penalty = 0.857 **Time = 0.024** Steps = 1	Coverage= 0. Penalty = 0 Time = 0 Steps = 0
Shallow UR + UR values Injection	Coverage=1 Penalty= 0.357 Time = 0.053 Steps=2	Coverage=1 Penalty= 0.428 Time = 614.83 Steps=10	Coverage=1 Penalty= 0.428 Time = 522.78 Steps=10	Coverage=1 Penalty= 0.795 Time = 10.98 Steps=2	Coverage=0.85 Penalty= 0.795 Time = 0.04 Steps=1	Coverage=0 Penalty= 0.0 Time = 0.002 Steps=0
Shallow UR + Targeted Noise Injection	Coverage=1 Penalty= 0.814 Time = 0.114 Steps=10	Coverage=1 Penalty= 0.799 Time = 486.99 Steps=10	Coverage=1 Penalty= 0.799 Time = 959.12 Steps=10	Coverage=1 Penalty= 0.811 Time = 32.24 Steps=2	Coverage=1 Penalty= 0.799 Time = 21.89 Steps=1	Coverage=0 Penalty=0 Time = 0.005 Steps=0

where OfflineCoverage variant on the *UR Values Injection* strategy sources, where coverage is achieved already using the first source but results in a higher penalty.

Generally, trained RL agents have a consistent and accurate choice of sources, achieving the maximum coverage possible in all cases (with OfflineAll serving as our reference for optimality). In terms of penalty, agents demonstrate robust minimization, particularly when evaluated on source splits using the same strategy as the one on which they are trained. For instance, RLValueInjection achieves near-optimal results with the *UR Value Injection* source strategy. This occurs even though its training was less stable than other variants. An issue occurs on the Shallow *UR* when training with RLTgtSourceInjection: as the agent has learned to use 0 sources, no source is selected and hence the results are 0 on all dimensions (coverage, penalty, sources). One way to mitigate this would be to use fewer episodes; indeed, the results seem to indicate that around 3000 episodes are enough for all agents for this type of *UR*. Finding the optimal number of episodes will be investigated in future work.

Additionally, we note that offline approaches tend to require more steps (i.e., sources selected) and longer evaluation time, as they lack a learned stopping mechanism and often require going through all of the available sources. RL agents, in contrast, often know when to terminate directly after finding a reasonable solution, reflecting effective policy learning.

This confirms the appropriateness of RL solution to solve the newly introduced TupleValDisc problem.

5 CONCLUSION AND FUTURE WORK

We formalized TupleValDisc, a novel dataset discovery problem that aims to build a target table from source tables. We propose a sequential decision making solution to solve TupleValDisc and we empirically explore different settings of source tables that validate our multi-objective formalization and opens several new directions.

Our immediate action is to examine other transfer learning settings. In our experiments, we reported the result of transferring from one set of source tables to another and saw that they are promising. We plan to study transfer across user requests and across datasets. Indeed, training an agent for a given request or on a given dataset could benefit other settings and prevent another training.

Our next action is to account for statistics of input tables. In our setup, we considered only equi-size source tables allowing us to focus on the number and order of those tables. Considering statistics such as table sizes and attribute selectivity in the definition of states and reward, would enrich the training of our agents.

Another direction is to define a ground truth for tuple-value discovery. While we use the offline method as a reference, we do not guarantee its optimality.

Following recent work [24], our next endeavor is to consider a language model as a data source. In our current setup, the agent selects from a fixed set of known sources that are locally available. A proposed future direction is to expand the agent's capabilities by introducing a new action: `Construct_Prompt`. When this action is selected, a prompt is dynamically generated based on the user request and previously selected sources, and sent to a pretrained language model. The goal is to retrieve external, potentially relevant

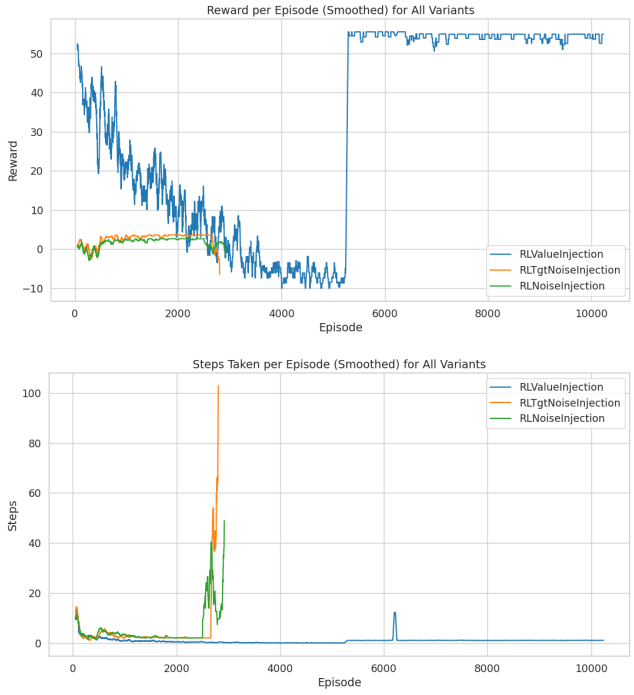


Figure 5: Training evolution with Ignore_Source_Repetition variant: reward (top) and steps (bottom).

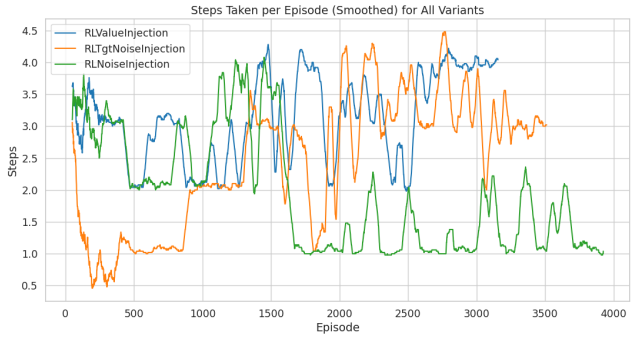


Figure 6: Steps per episode for each RL agent with No-Step-Weight

tabular data from online sources that can contribute to improving coverage. This would require to revisit our reward to include the cost of using a language model as a data source.

ACKNOWLEDGMENTS

This work was supported by DataGEMS, funded by the European Union's Horizon Europe Research and Innovation programme, under grant agreement No 101188416.

REFERENCES

[1] Alex Bogatu, Alvaro AA Fernandes, Norman W Paton, and Nikolaos Konstantinou. 2020. Dataset discovery in data lakes. In *2020 ieee 36th international conference on data engineering (icde)*. IEEE, 709–720.

[2] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. 2009. Data integration for the relational web. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1090–1101.

[3] Peter Baile Chen, Yi Zhang, and Dan Roth. 2024. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. *arXiv preprint arXiv:2404.09889* (2024).

[4] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D Davison. 2020. Table search using a deep contextualized language model. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. 589–598.

[5] Arash Dargahi Nobari and Davood Rafiei. 2024. Dtt: An example-driven tabular transformer for joinability by leveraging large language models. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–24.

[6] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. 2017. Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I 16*. Springer, 260–277.

[7] Grace Fan, Roee Shraga, and Renée J. Miller. 2024. Gen-T: Table Reclamation in Data Lakes. arXiv:2403.14128 [cs.DB] https://arxiv.org/abs/2403.14128

[8] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée Miller. 2023. Semantics-Aware Dataset Discovery from Data Lakes with Contextualized Column-Based Representation Learning. *Proceedings of the VLDB Endowment* 16 (05 2023), 1726–1739. https://doi.org/10.14778/3587136.3587146

[9] Yukihisa Fujita, Teruaki Hayashi, and Masahiro Kuwahara. 2024. Inferring Relationships between Tabular Data and Topics using LLM for a Dataset Search Task. In *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 6564–6573.

[10] Yuan He, Jiaoyan Chen, Denvar Antonyrajah, and Ian Horrocks. 2022. BERTMap: a BERT-based ontology alignment system. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 5684–5691.

[11] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and Philip S Yu. 2023. Automatic table union search with tabular representation learning. In *Findings of the Association for Computational Linguistics: ACL 2023*. 3786–3800.

[12] Paras Jain, Xin Luna Dong, Arash Nargesian, and Michael Stonebraker. 2023. Large Language Models for Data Discovery and Integration: Challenges and Opportunities. *IEEE Data Eng. Bull.* 46, 1 (2023), 3–14. http://sites.computer.org/debull/A23mar/p3.pdf

[13] Nikolaos Kardoulakis, Kenza Kellou-Menouer, Georgia Troullinou, Zoubida Kedad, Dimitris Plexousakis, and Haridimos Kondylakis. 2021. Hint: Hybrid and incremental type discovery for large RDF data sources. In *Proceedings of the 33rd International Conference on Scientific and Statistical Database Management*. 97–108.

[14] Kaiwen Li, Tao Zhang, and Rui Wang. 2020. Deep Reinforcement Learning for Multi-objective Optimization. *IEEE Transactions on Cybernetics* 50, 12 (2020), 4915–4926. https://doi.org/10.1109/TCYB.2020.2977661

[15] Xinyun Li, Pengcheng Yin, Xilun Chen, Pranav Raghavan, Oleksandr Polozov, Matthew Richardson, Duyu Wang, Wen-tau Yih, Preksha Nema Jain, Zhiguo Guo, Muhao Chen, and Mu Li. 2023. ELEET: Efficient Learned Query Execution over Text and Tables. In *Proceedings of the 2023 ACM SIGMOD International Conference on Management of Data (SIGMOD '23)*. ACM, 1809–1822. https://doi.org/10.1145/3588945.3597444

[16] Junfei Liu, Shaotong Sun, and Fatemeh Nargesian. 2024. Causal Dataset Discovery with Large Language Models. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*. 1–8.

[17] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825.

[18] Phuc Nguyen, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. 2019. Mtab: Matching tabular data to knowledge graph using probability models. *arXiv preprint arXiv:1910.00246* (2019).

[19] Masayo Ota, Heiko Mueller, Juliana Freire, and Divesh Srivastava. 2020. Data-driven domain discovery for structured datasets. *Proceedings of the VLDB Endowment* 13, 7 (2020), 953–967.

[20] Masayo Ota, Heiko Müller, Juliana Freire, and Divesh Srivastava. 2020. Data-Driven Domain Discovery for Structured Datasets. *Proceedings of the VLDB Endowment* 13, 7 (2020), 953–966. https://doi.org/10.14778/3384345.3384352

[21] Norman W. Paton, Jiaoyan Chen, and Zhenyu Wu. 2024. Dataset Discovery and Exploration: A Survey. *ACM Comput. Surv.* 56, 4 (2024), 102:1–102:37.

[22] Federico Piai, Paolo Atzeni, Paolo Merialdo, and Divesh Srivastava. 2023. Fine-grained semantic type discovery for heterogeneous sources using clustering. *The VLDB Journal* 32, 2 (2023), 305–324.

[23] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web using Column Keywords. *Proceedings of the VLDB Endowment* 5 (06 2012). https://doi.org/10.14778/2336664.2336665
[24] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. 2024. Querying Large Language Models with SQL. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*, Letizia Tanca, Qiong Luo, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani (Eds.). OpenProceedings.org, 365–372.
[25] Aécio Santos, Aline Bessa, Christopher Musco, and Juliana Freire. 2022. A sketch-based index for correlated dataset search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2928–2941.
[26] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized Experience Replay. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1511.05952
[27] Levy Silva and Luciano Barbosa. 2024. Improving dense retrieval models with LLM augmented data for dataset search. *Knowledge-Based Systems* 294 (2024), 111740.
[28] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3 (1988), 9–44.
[29] Yuroti Tsuboi and Nobutaka Suzuki. 2019. An algorithm for extracting shape expression schemas from graphs. In *Proceedings of the ACM Symposium on Document Engineering 2019*. 1–4.
[30] Qiming Wang and Raul Castro Fernandez. 2023. Solo: Data Discovery Using Natural Language Questions Via A Self-Supervised Approach. *Proceedings of the ACM on Management of Data* 1, 4 (2023), 1–27.
[31] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 97–108.
[32] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. 2019. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation. *CoRR* abs/1908.08342 (2019).
[33] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. 2022. Cost-based or learning-based? A hybrid query optimizer for query plan selection. *Proceedings of the VLDB Endowment* 15, 13 (2022), 3924–3936.
[34] Shuo Zhang and Krisztian Balog. 2017. Entitables: Smart assistance for entity-focused tables. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*. 255–264.
[35] Shuo Zhang and Krisztian Balog. 2018. Ad hoc table retrieval using semantic similarity. In *Proceedings of the 2018 world wide web conference*. 1553–1562.
[36] Yi Zhang and Zachary G Ives. 2020. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1951–1966.

A DETAIL OF USER REQUESTS

Deep UR (3 attributes, 4 values each):

- `keyword_name`: Two variables, Orthogonality, Three points rule, Mean
- `topic_name`: Linear Algebra, Probability, Optimization, Discrete Mathematics
- `subtopic_name`: Linear Transformations, Vector Spaces, Algebraic expressions, Equations, and Inequalities, Triple Integration

Shallow UR (7 attributes, mostly 1 value each):

- `keyword_name`: Cauchy problem
- `topic_name`: Integration, Discrete Mathematics
- `subtopic_name`: Recursivity
- `question_id`: 80
- `id_lect`: 2162
- `answer1`: The system has no solution.
- `keyword_id`: 139