LakeBench: A Benchmark for Discovering Joinable and Unionable Tables in Data Lakes

Yuhao Deng
dyh18@bit.edu.cn
Beijing Institute of
Technology

Chengliang Chai
ccl@bit.edu.cn
Beijing Institute of
Technology

Lei Cao
lcao@csail.mit.edu
University of
Arizona/MIT

Qin Yuan
yuanq1020@gmail.com
Beijing Institute of
Technology

Siyuan Chen
siyuanchen@bit.edu.cn
Beijing Institute of
Technology

Yanrui Yu
yanruiyu@bit.edu.cn
Beijing Institute of
Technology

Zhaoze Sun
sunzhaoze@bit.edu.cn
Beijing Institute of
Technology

Junyi Wang
wangjunyi@bit.edu.cn
Beijing Institute of
Technology

Jiajun Li
lijiajun@bit.edu.cn
Beijing Institute of
Technology

Ziqi Cao
caoziqi@bit.edu.cn
Beijing Institute of
Technology

Kaisen Jin
jks@bit.edu.cn
Beijing Institute of
Technology

Chi Zhang
zhangchi315@bit.edu.cn
Beijing Institute of
Technology

Yuqing Jiang
jiangyuqing@bit.edu.cn
Beijing Institute of
Technology

Yuanfang Zhang
yfzhang@bit.edu.cn
Beijing Institute of
Technology

Yuping Wang
wyp_cs@bit.edu.cn
Beijing Institute of
Technology

Ye Yuan
yuan-ye@bit.edu.cn
Beijing Institute of
Technology

Guoren Wang
wanggr@bit.edu.cn
Beijing Institute of
Technology

Nan Tang
nantang@hkust-
gz.edu.cn
HKUST (GZ)

ABSTRACT

Discovering tables from poorly maintained data lakes is a significant challenge in data management. Two key tasks are identifying joinable and unionable tables, crucial for data integration, analysis, and machine learning. However, there's a lack of a comprehensive benchmark for evaluating existing methods. To address this, we introduce LakeBench, a large-scale table discovery benchmark. It evaluates effectiveness, efficiency, and scalability of table join & union search methods. With over 16 million real tables, LakeBench is **1,600X** larger than existing datasets and **100X** larger in storage size. It includes synthesized and real queries with ground truth, totaling more than 10 thousand queries – **10X** more than used in any existing evaluation. We spent over 7,500 human hours labeling these queries and constructing diverse query categories for thorough evaluation. Our benchmark thoroughly evaluates state-of-the-art table discovery methods, providing insights into their performance and highlighting research opportunities.

1 INTRODUCTION

Many governments have acknowledged the importance of providing open access to their data for the public good [30]. However, these open datasets are typically published in formats like image or JSON that don't fit neatly into relational databases. To make sense of this poorly maintained data, data scientists have to go through a pipeline of data curation tasks to discover, merge, and clean data [5, 9–11, 31]. As the first step of this pipeline, data discovery that finds relevant tables is critical for data scientists to effectively and efficiently conduct their work.
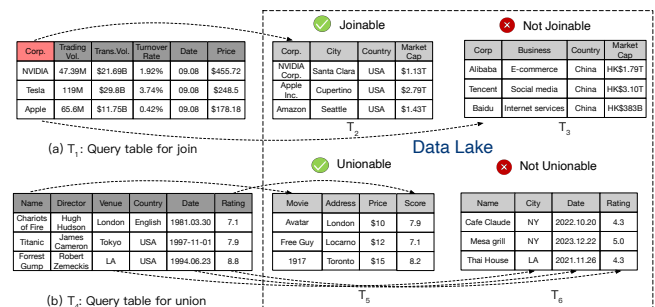

Figure 1: Table Discovery in Data Lake.

Table discovery from data lakes can be generally classified into two categories: keyword-based search [4, 16, 19, 20, 40] and table-based search [3, 13, 14, 17, 24, 32, 45, 46]. Keyword-based search over

tables often relies on the metadata of tables in the lake to identify tables or tuples relevant to the given keywords. However, in the data lake scenario, the metadata is often poorly maintained compared to the database management system (DBMS). Consequently, the incompleteness and inconsistency of metadata in the data lake become bottlenecks for keyword-based search. Thus in this work, we focus on the table-based search that can be further categorized into table join search and table union search. Recently, table-based search has attracted much attention in data management field to augment data instances [7, 27, 43] or attributes [6, 8, 26, 28, 39]. This benefits many downstream applications such as machine learning based data analysis.

Existing Works. To the best of our knowledge, no work has specifically focused on constructing a benchmark for the thorough evaluation of the existing table search methods. Although some works that target inventing new table search methods have done a great job in releasing their datasets and queries used in their evaluation, the scale of their datasets, the number of queries, and the diversity of the workload are not sufficient to evaluate the effectiveness, scalability, and efficiency of the table search methods in diverse scenarios. In particular, TUS [32] and Santos [24] evaluate their *table union search* methods with 1,000 queries on *small* data lakes which contain at most 11,090 tables – in total 11GB data, far from being sufficient to evaluate the scalability of the search algorithms and simulate the diverse real applications. Josie [45], targeting *table join search*, releases 1,000 queries to evaluate if their method is able to efficiently find the tables that have exact overlap with the query tables. In this setting, they do not have to label their queries, e.g., manually discover the tables joinable with the query tables. Therefore, these queries cannot be used to evaluate the effectiveness of other table join search methods that consider the semantic similarity of the data.

Design Goals. To fill this gap, in this work we aim to construct a comprehensive benchmark for table discovery, following the design goals below:

(1) *Scalability.* To thoroughly evaluate the efficiency and scalability of different methods, the benchmark has to involve data lakes of various sizes, especially large-scale ones. The size of the data lake is measured from two aspects: the storage size and the number of total columns. The storage size is highly related to the average table size and the number of tables. The number of total columns matters as well because almost all table discovery algorithms index and search over a large number of columns.

(2) *Variety.* The performance of table discovery largely relies on the characteristics of the datasets and queries. In particular, one algorithm might perform differently on different categories of queries. For example, given a query, a semantic-aware algorithm will show clear advantages if the table semantic matters the most in determining its unionable tables. Hence, plenty of diverse queries are critical to thorough evaluation.

(3) *Comprehensive comparison of solutions.* Understanding the pros/cons of existing table search methods and revealing research opportunities, requires a reliable implementation of all these methods, a thorough and fair experimental evaluation, and a deep analysis of the results.

Our Proposal. Guided by the above three design goals, we construct a benchmark, LakeBench– the first of its kind that supports the thorough evaluation of table union & join methods.

To achieve the first design goal, we collect over 1TB data – **100X** larger than the data lakes used in existing works, and the number of columns is up to **100 million**, on which we evaluate different table union/join search methods. Moreover, we label a sufficient number of diverse queries (more than 10 thousand) – **10X** more than any other works. To this end, we organize a team of 25 graduate students and spent more than *7,500 human hours* on labeling the queries. Because it is prohibitively expensive to manually identify tables in the data lake that are joinable/unionable to a given query, we design a candidate generation strategy to prune the tables that are most unlikely to be the ground truth.

To meet the second design goal, we construct data lakes with diverse characteristics by collecting tables from two data sources, namely OpenData [1] and WebTable [2]. WebTable has a large number of tables (more than 10^8) but each table is small. In contrast, OpenData consists of large tables (on average 1GB in size per table) but has fewer tables than WebTable. Besides, we generate both synthetic (splitting from large lake tables) and real queries (directly sampling from the data lake), which have different properties. We further categorize these queries at finer granularity w.r.t. the key factors that determine the quality of table search.

For the third design goal, we implement many key techniques such as column embeddings, approximate nearest neighbor (ANN) search index, etc., which are the building blocks of these approaches, ensuring that we are able to evaluate the scalability and accuracy of various approaches on different datasets and mostly reuse the code base. More importantly, we evaluate these approaches over the aforementioned fine-grained query categories, so as to reveal multiple insights w.r.t. the performance of different approaches over different categories. In addition, our analysis of the time and space complexity of the search algorithm connects the experiment results to theory, guiding our discussions on the results. In short, LakeBench makes it possible to thoroughly evaluate and understand existing table search methods.

Contributions. We make the following contributions:

(1) We build a comprehensive benchmark, LakeBench, for table discovery, featuring large-scale datasets and a sufficient number of diverse queries. Besides, it addresses critical limitations in existing benchmarking datasets. To be specific, *i.*) The lack of a specific dataset for joinable search with ground truth; *ii.*) The number of ground truth tables for individual queries is small, particularly notable in TUS and SANTOS, thus problematic in accurately assessing recall; *iii.*) Relying heavily on automatically generated ground truth, which may simplify the task and potentially miss ground truth because of lacking sufficient human annotation.

(2) We collect real tables (more than 1 TB in size) from multiple sources, which consist of 4 data lakes with various storage sizes and different numbers of table columns.

(3) We create 10,000+ table queries covering different characteristics and spend 7,500+ human hours on accurately labeling them.

(4) We thoroughly evaluate and analyze the state-of-the-art table join/union search approaches using these created queries on the benchmark datasets. Based on that, we reveal multiple insights and research opportunities. For example, we observe that although

the methods that fine-tune pre-trained language models in general show an advantage over other methods, their accuracy is still not very satisfactory in some cases. Therefore, designing better fine-tune strategies or fine-tuning large language models (LLM) might be a promising direction to go in the future.

2 PRELIMINARY

2.1 Problem Definition

We formally define the two table discovery tasks in a data lake.

Table Join Search. Suppose that a data lake $\mathcal{T}$ contains a large set of tables $\mathcal{T} = \{T_1, T_2, ..., T_N\}$, where each table $T_i, i \in [1, N]$ has n_i rows (tuples), m_i columns (attributes) and each cell value is denoted by c_{ij}. Given a query table T_q^J, as well as a specific column C_q^J of T_q^J, table join search is to find the target tables that can be joined with T_q^J on C_q^J, and we note the measure of the relevance score between two columns as $R(C_q^J, C_t)$, where C_t is a column of a target table $T_t \in \mathcal{T}$. The higher the score, the more likely the two columns can be joined (In this case, we can say that the two columns or two tables can be joined interchangeably), *i.e.*, $R(C_q^J, C_t)$ can also be represented as $R(C_q^J, T_t)$. Note that different methods have different criteria to compute the score, like the number of overlaps and/or semantic similarity. The formal definition is as follows.

Definition 1 (Top-k Table Join Search). Given $\mathcal{T}$, a query table T_q^J, the specific column C_q^J and a parameter k, top-k table join search aims to retrieve a subset $\mathcal{T}_q \subset \mathcal{T}$, $|\mathcal{T}_q| = k$ such that $\forall T \in \mathcal{T}_q$ and $\forall T' \in \mathcal{T} \setminus \mathcal{T}_q, R(C_q^J, T) > R(C_q^J, T')$.

Remark. Given a target table T_t, there may exist multiple columns that can be joined with T_q. In this case, we take the one with the highest score as the target column, *i.e.*, $C_t^J = \arg\max_{C \in T_t} R(C_q^J, C)$. Also, we focus on the two-way join rather than multi-way joins [15].

Table Union Search. Given a query table T_q^U, table union search aims at finding the top-k unionable tables from the data lake $\mathcal{T}$. At a high level, similar to table join search, table union search highly relies on the unionbility of the columns, *i.e.*, a pair of unionable tables should have multiple pairs of columns that can be unioned. Like table join search, column unionbility also takes the overlaps and/or semantics between columns into consideration. To be specific, given a target table T_t, we can measure the relevance of each pair of columns, *i.e.*, $R(C, C'), C \in T_q^U, C' \in T_t$, and compute a table-level relevance score as $R(T_q^U, T_t)$.

Definition 2 (Top-k Table Union Search). Given $\mathcal{T}$ and a query table T_q^U, top-k table union search aims to retrieve a subset $\mathcal{T}_q \subset \mathcal{T}$, $|\mathcal{T}_q| = k$ such that $\forall T \in \mathcal{T}_q$ and $\forall T' \in \mathcal{T} \setminus \mathcal{T}_q, R(T_q^U, T) > R(T_q^U, T')$.

More specifically, given a query table with a user-specified column, table join search finds the target tables that can be joined with the query table on the user-specified column.

EXAMPLE 1 (TABLE JOIN SEARCH). *As shown in Figure 1(a), given a query table (T_1) and a specified column* Corporation, *Table T_2 in the data lake is joinable with T_1, i.e., the first attribute of T_2 can be joined with the specified column because i) the column names match and ii) a number of cell values have fuzzy overlaps (e.g., Apple and Apple*
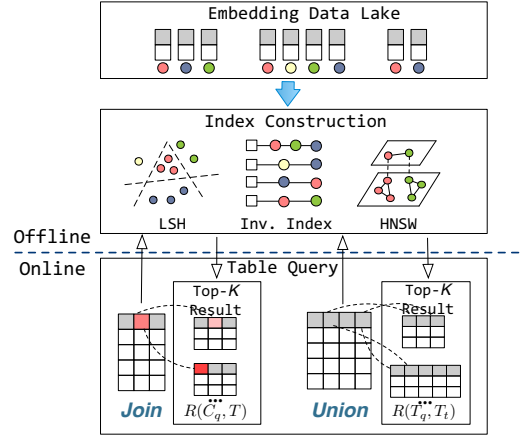

Figure 2: Table Discovery in Data Lake.

Inc.). Joining these two tables together produces a table with richer features, which is beneficial when used to train a machine learning model to for example predict the stock Price *of the companies in the table. On the other hand, although in T_3, the column name and the contents of its first attribute are semantically similar to the specified column, it is not joinable because there is no overlapping value.*

For table union search, given a query table, it aims to find target tables that are unionable with the query.

EXAMPLE 2 (TABLE UNION SEARCH). *As shown in Figure 1(b), given a query table (T_4), T_5 can be unioned with T_4 because they are all about the information of movies and two pairs of attributes are highly correlated (T_4.Name v.s. T_5.Movie and T_4.Rating v.s. T_5.Score). However, when it comes to T_6, although T_6 has three attributes (T_6.City, T_6.Date and T_6.Rating) aligned respectively with attributes (T_4.Venue, T_4.Date and T_4.Rating) of T_4, T_6 is not unionable with T_4. This is because the two tables are not in the same context: T_4 is about movies, while T_6 is about restaurants.*

Examples 1 & 2 show that effectively discovering joinable and unionable tables faces multiple common challenges, including (1) the (semantic) schema similarity of columns, (2) the overlap between columns, and (3) the contextual information across multiple columns in a table. Moreover, as a data lake is typically large-scale, designing efficient and scalable discovery algorithms is of high complexity. Although many approaches have been proposed to address the above problem from different perspectives, these approaches are yet to be thoroughly evaluated due to the lack of a comprehensive benchmark.

Remark. Note we return the top-k results rather than asking users to set a cut-off threshold w.r.t. the relevance score and returning all tables that have a relevance score higher than this threshold. This is because setting a threshold appropriate to the given query is harder than setting the k parameter. Therefore, almost all [3, 14, 17] existing works focus on retrieving the top-k results.

2.2 Table Discovery Process

Most of the table discovery methods consist of the following three modules: data lake embedding, index construction, and online table

Table 1: Statistics of Data Lakes.

Data Lake	#-Max/Min/Avg Col.	#-Max/Min/Avg Row
OpenData	502 / 3 / 16.1	10,250,220 / 5 / 79,310.7
OpenData Large	502 / 3 / 21.0	39,014,091 / 5 / 112,359.9
WebTable	25 / 3 / 6.5	16,908 / 5 / 23.0
WebTable Large	25 / 3 / 6.8	16,908 / 5 / 23.5

query processing, as shown in Figure 2. The former two modules are offline, which respectively encode the column/schema in the data lake into a vector and index these vectors. Once an online query table arrives, we first encode its column(s) and then use the index to retrieve the top-k tables with the highest relevance score from the data lake. Next, we illustrate each module respectively.

Embedding Data Lake. In $\mathcal{T}$, original table columns are consistently encoded as fixed-length vectors (as detailed in Section 2). The vectors are mostly hash codes [18, 46] (*e.g.*, generated by the minhash function) or embeddings [17, 21, 44] (*e.g.*, generated by pre-trained language models) that are effective in finding columns with similar semantics or high overlaps. However, methods like Josie and InfoGather [41, 45] directly use the original cell values of each column to search highly overlapping columns rather than using vectors.

Index Construction. Almost all table discovery methods rely on an index to search large table repositories. With column representations, common choices for ANN indexes include Local Sensitive Hashing (LSH) [3, 46] or Hierarchical Navigable Small World (HNSW) [14, 17]. Alternatively, an inverted index [24, 41, 45] can expedite finding highly overlapping columns by linking cell values to containing columns. Here, colorful circles denote Column IDs instead of vectors.

Online Table Discovery. Once a user issues a table T_q^J for *join* search, the search algorithm first represents C_q^J as a vector. It then uses the index to quickly identify the top-k columns (tables) with high relevance scores, *i.e.*, $R(C_q^J, T_t)$.

The *union* search follows the same procedure. The only difference is that given a table T_q^U, the union search has to search over each attribute in T_q^U and aggregate the results. More specifically, suppose that T_q^U has m columns. For each column $C_i \in T_q^U, i \in [1, m]$, similar to the join search, we retrieve from the data lake a set of columns that have high relevance scores with C_i. Then, we compute the union of all retrieved tables, *i.e.*, $\cup_{i=1}^m C_i$. For each table $T_t \in \cup_{i=1}^m C_i$, the relevance score $R(T_q^U, T_t)$ is computed using techniques like maximum bipartite graph matching, considering the column relevance between the two tables. Finally, the top-k tables with the highest relevance scores are returned.

3 BENCHMARK DESIGN

We first overview the process of building LakeBench, and introduce our efforts on labeling the queries.

3.1 Benchmark Construction Process

As shown in Figure 3, we first collect the datasets. Then we construct and label queries. Finally, we implement various algorithms and analyze their performance from multiple perspectives.
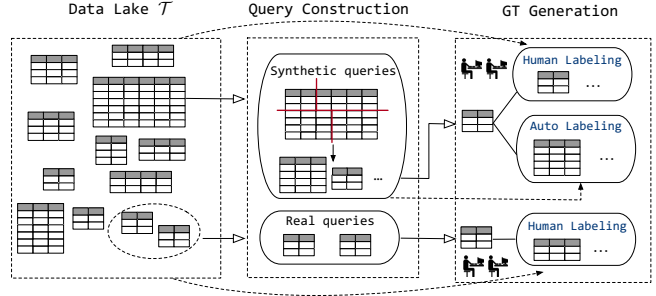

Figure 3: Overview of LakeBench Construction.

Datasets. We build 4 data lakes from two data sources: Open-Data [1] and WebTable [2]. From each source, we create two data lakes in different sizes. Table 1 summarizes the statistics of these data lakes. We extract 64,698 tables from the Open Data portals in Canada, the UK, the USA, and Singapore and derive OpenData Large. For WebTable, we pre-process 16,670,064 tables into the data lake after removing extremely small tables and null columns, producing WebTable Large. Finally, we randomly sample 10% from OpenData Large and 17% from WebTable Large to construct OpenData and WebTable respectively.

Query Construction. We construct join & union search queries in two ways.

Real queries. One way is to directly use real tables as queries, so we extract $Q \subset \mathcal{T}$ as a set of queries to simulate the table search needs in the real world. Note that real queries show diverse characteristics. Therefore, we further classify them into finer-grained query categories and evaluate the performance of the search algorithms in diverse scenarios (Section 5.3).

Synthetic queries. However, in the above real cases, table discovery results are likely to be sparse – each query gets only several joinable/unionable tables. Hence, all previous works construct synthetic queries; and we do this similarly as follows. The basic idea is to split large tables into multiple small ones. In Figure 3, we split large tables at either row-level or column-level. Generally, splitting at row-level mainly generates unionable tables, while splitting at column-level mostly generates joinable tables. These small tables are put into the data lake and served as queries. Next, we introduce the more concrete process of constructing synthetic queries.

[Choose large tables:] We prefer large base tables with more rows and columns to split. To achieve this, we will pick a table only if the numbers of its row and columns are both larger than a threshold (*i.e.*, for Opendata, we set it as 50; for Webtable, we set it as 20). We then sort these tables based on the number of cells and select the top-20% largest tables to split.

[Synthesize join queries:] To be joinable, a pair of tables should contain at least one column on which they could join; and the two tables should overlap substantially on the cell values of this column. To achieve this, we first randomly select from a large table a column as the joinable column, and split the large table vertically into two sub-tables that share the selected column. Second, to create the overlaps, we randomly select some rows shared between the two sub-tables. Third, we separately choose a set of rows from the two sub-tables; together with the overlapping rows, each set forms

a new sub-table. The two new sub-tables thus can join with each other and serve as join queries. Besides, followed Josie [45], we only focus on non-numerical data as numerical values tend to generate casual joins that lack meaningful significance.

[Synthesize union queries:] A pair of unionable tables typically is expected to share some columns from the same domain. Consequently, to synthesize queries from a large table, we first randomly select multiple columns as the shared columns and then split the large tables horizontally into multiple sub-tables. Second, in each sub-table, together with the shared columns, we randomly select several other columns as supplementary columns to produce a synthetic union query. In this way, the query tables synthesized from each sub-table are unionable because of these shared columns.

Labeling Queries. We design different labeling methods w.r.t. queries constructed in different ways, as shown in Figure 3.

Labeling real queries. Given a query, to generate high-quality ground truth, the ideal way is to ask the humans to examine each table in $\mathcal{T}$ and see if it is joinable/unionable with the given query. However, this is prohibitively expensive. So we leverage a table retrieval method to return a set of candidate tables. Then we ask the humans to manually verify these candidates.

Labeling synthetic queries. As shown in Figure 3, the ground truth of synthetic queries mainly comes from two sources: (1) the tables that are produced from the same big tables as the query table during splitting; (2) some other tables that are derived from the original tables joinable/unionable with the given query table or its original table. It is straightforward to identify the first source, while we can leverage the above method that labels real queries to identify the second source.

3.2 Human Labeling

To label the ground truth of a query, we first generate a number of candidates likely to be joinable/unioable with the query, and then ask the experts to label them with the possibility to early stop.

Candidates Generation. For each synthetic or real query, we design a search strategy to retrieve a set of candidate tables joinable/unionable with a query, which will be manually examined by human experts. The goal is two-fold: (1) to achieve a high recall, *i.e.*, given a query, a comprehensive set of joinable/unionble tables is included among the candidate tables; (2) to minimize the human efforts.

Ensemble Retrieval. At a high level, to achieve a high recall, we propose to leverage multiple typical table discovery methods to retrieve a number of candidates and ensemble them as the final candidate set. To ensure a high recall, we set a relatively large K for candidate generation and display them to the experts for labeling.

Early Stopping. However, given thousands of queries, asking human experts to check all the above candidates is still rather expensive. Therefore, we propose a simple but effective early stopping strategy to reduce human efforts. Specifically, since the provided list of candidates is ordered, it is reasonable to assume that if a candidate is not unionable, its subsequent candidates are likely not unionable. Therefore, we ask the experts to label along the ordered list, and if among 10 successive tables, fewer than 20% (*i.e.*, 2 tables) are unionable, they will stop and jump to the next query to label. We

Table 2: Statistics of Human Labeling.

Data Lake	#-Join/Union Queries	#-Experts	Avg. Time
OpenData	3,171 / 3,095	25	31.6h / 36.1h
OpenData Large	4,762 / 4,580	25	47.6h / 53.4h
WebTable	5,824 / 5,487	25	25.9h / 30.5h
WebTable Large	7,428 / 6,812	25	33.0h / 37.9h

also build a platform to support the labeling process with a friendly user interface.

Labeling Statistics. We hire 25 graduate students from the database group in our university. All students are familiar with the table discovery task. Table 2 summarizes the labeling statistics. For example, for union queries of OpenData, we construct 2,171 synthetic queries by splitting 60 large tables and extract 1,000 real tables from the original data lake as real queries. So in total, we produce 3,171 queries. Subsequently, for union queries of OpenData Large, we additionally select 40 tables from OpenData Large but not in OpenData, which are then split into 1,091 synthetic queries. We sample 500 more tables as real queries. Next, we detail the work of human experts. Take the OpenData Large as an example, which has 4,762 queries. For each query, on average 30 candidate tables are labeled by experts. Each expert spends approximately 47.6 hours on labeling. In total, we consume about 7,500 human hours on labeling.

4 TABLE JOIN AND UNION METHODS

In this section, we illustrate approaches in the three categories, which are classified based on the claims of the papers.

4.1 Table Join Search

According to [32], we further classify join search approaches into two categories, namely set domain [45, 46] and natural language domain [13, 14], according to their techniques.

Set domain. Josie [45] targets finding joinable tables via *set similarity search*. Based on the intuition that two joinable columns should have many overlapping values, given a query table T_q and query column C_q, Josie considers C_q as a set, and returns the k columns that have the highest overlap with C_q, measured as set similarities. Josie builds an inverted index to optimize the search speed. It maps distinct cell values to sets (columns, namely posting lists) that contain the corresponding values. Then given a query column, it uses the inverted index to search candidate columns with overlapping values. It introduces a cost model to quickly eliminate unqualified candidates. The time complexity of building the index is $O(C + \mathcal{R} \log \mathcal{R})$, relating to the number of distinct values in $\mathcal{T}$, while the online time complexity is $O(\mathcal{L} \log \mathcal{L})$, which is influenced by the length of posting list.

Similar to Josie, LSH Ensemble [46] also considers column set overlap. However, instead of computing exact top-k columns, it estimates overlap using the set containment score derived from the MinHash LSH index. Columns exceeding a threshold score are returned. After retrieving these columns, we further rank the results based on the overlap with the query column, resulting in the top-k output. To enhance efficiency, columns are initially partitioned based on their lengths, reducing the need for an online query to compare with all lake columns. Second, several MinHash LSH indexes are constructed for each partition. Both the offline and

Table 3: Table Discovery Methods.

Methods	Task	Index	Embedding	Offline Complexity		Online Complexity	
				Time	Space	Time	Space
Josie [45]	J	Inv. index	✗	$O(C + \mathcal{R}\log\mathcal{R})$	$O(\mathcal{R})$	$O(\mathcal{L}\log\mathcal{L})$	$O(\mathcal{L})$
LSH Ensemble [46]	J	LSH	✗	$O(\mathcal{N}\mathcal{H}\mathcal{V})$	$O(\mathcal{D}\mathcal{N}\mathcal{H}\mathcal{V})$	$O(\mathcal{B}\mathcal{V}\mathcal{H})$	$O(\mathcal{B})$
Pexeso [13]	J	Inv. index	✓	$O(\mathcal{R})$	$O(\mathcal{R})$	$O(\log\mathcal{A}\log\mathcal{R})$	$O(\mathcal{A})$
DeepJoin [14]	J	HNSW	✓	$O(\mathcal{N}\log\mathcal{N})$	$O(\mathcal{D}\mathcal{N})$	$O(\log\mathcal{N})$	$O(\mathcal{N})$
TUS [32]	U	LSH	✓	$O(C + \mathcal{N}\mathcal{H}\mathcal{V})$	$O(\mathcal{D}\mathcal{N}\mathcal{H}\mathcal{V})$	$O(d^3\mathcal{S})$	$O(\mathcal{B}d^2\mathcal{S})$
D3L [3]	U	LSH	✓	$O(C + \mathcal{N}\mathcal{H}\mathcal{V})$	$O(\mathcal{D}\mathcal{N}\mathcal{H}\mathcal{V})$	$O(\mathcal{B}\mathcal{E})$	$O(\mathcal{B})$
Santos [24]	U	Inv. index	✗	$O(n^2m\mid\mathcal{T}\mid)$	$O(n^3m\mid\mathcal{T}\mid^2)$	$O(n^2m)$	$O(\mathcal{A})$
Starmie [17]	U	HNSW	✓	$O(\mathcal{N}\log\mathcal{N})$	$O(\mathcal{D}\mathcal{N})$	$O(\log\mathcal{N})$	$O(\mathcal{N})$
Frt12 [36]	J & U	N/A	✗	$O(\mathcal{N})$	$O(\mathcal{N})$	$O(\mid\mathcal{T}\mid\cdot(\mathcal{B}+\overline{O})^3)$	$O(\overline{O}^2)$
InfoGather [41]	J & U	Inv. index	✗	$O(\mathcal{R} + \mathcal{K}\mathcal{N})$	$O(\mathcal{R})$	$O(\mathcal{B}\mathcal{I}\log\mathcal{I})$	$O(\mathcal{I})$
Aurum [18]	J & U	LSH	✓	$O(\mathcal{N}\mathcal{H}\mathcal{V})$	$O(\mathcal{D}\mathcal{N}\mathcal{H}\mathcal{V})$	$O(\mathcal{B}\mathcal{V}\mathcal{H})$	$O(\mathcal{B})$

Table 4: Notations used in Table 3.

Notation	Explanation
$\mathcal{B}$	#-Columns in the query table T_q
$\mathcal{N}$	#-Columns of all tables in $\mathcal{T}$
C	#-Cell values of all tables in $\mathcal{T}$
$\mathcal{R}$	#-Distinct cell values of all tables in $\mathcal{T}$
$\mid\mathcal{T}\mid$	#-Tables in $\mathcal{T}$
$\mathcal{X}$	#-Tuples of all tables in $\mathcal{T}$
$\mathcal{A}$	#-Cell values in query table T_q
$\mathcal{L}$	Length of posting list in Josie
$\mathcal{H}$	Bucket size for each hash table
$\mathcal{V}$	#-Hash tables
$\mathcal{H}$	#-Hash functions
$\mathcal{P}$	#-Partitions in LSH Ensemble
$\mathcal{E}$	#-Neighbors in D3L
$\mathcal{D}$	Dimension of embeddings
$\mathcal{I}$	#-Neighbors in InfoGather
$\mathcal{S}$	#-Tables retrieved by LSH index in TUS
n	The largest #-columns in a table of $\mathcal{T}$
m	The largest #-rows in a table of $\mathcal{T}$
$\overline{O}$	Average #-columns in candidate tables

online time complexities are linear to the number of columns in $\mathcal{T}$ (T_q).

Natural language domain. Rather than computing the overlaps, natural language domain methods consider the semantic similarity between columns. These methods encode the columns, index these columns, and search join pairs based on the index. Their differences are on the encoding methods and the types of indexes. Pexeso [13] first encodes the cell values into high-dimensional vectors using fastText [23], which are then indexed with an inverted index and a hierarchical grid. As a query C_q comes, a block-and-verify strategy is applied to efficiently compute the cosine similarity between the vectors. The online time complexity is $O(\log\mathcal{A}\log\mathcal{R})$, which is dominated by the number of distinct cell in $\mathcal{T}$, so it is hard to scale this method to large data lakes.

Different from Pexeso, DeepJoin fine-tunes the pre-trained language model DistilBERT [35] and MPNet [37]. It then feeds a pair of columns into the models, outputs two vectors and computes their cosine similarity. The difference between this similarity and the actual similarity of this column pair is used as the loss to train the model. Afterwards, DeepJoin embeds columns in $\mathcal{T}$ into vectors through model inference and indexes them using HNSW [29]. When C_q comes, DeepJoin transforms it into a vector on the fly and retrieves similar columns that are likely to be joinable through the HNSW index. The complexity of this method is directly tied to the construction and query of the HNSW index, as shown in Table 3.

Pros & Cons. The two categories of join search methods perform differently in different cases. In the real world, data is likely to be dirty, and thus a pair of cell values might not exactly match but in fact point to the same entity (*e.g.*, Apple and Apple Inc.), namely fuzzy overlaps. In this case, natural language domain methods will show advantages, because the embeddings they produce capture the semantic similarity. However, if two columns share similar domains but no (fuzzy) overlaps, natural language domain methods are likely to suffer from many false positives.

4.2 Table Union Search

All methods of union search consider the semantic information of columns or tables, and also we further classify them into column independent [3, 32] and contextual approaches [17, 24] refer to [32].

Column independent methods. Given a query and a candidate table, these methods form many pairs of columns, where each column in a pair comes from a different table. Then these methods independently evaluate whether each column pair is unionable, without considering their correlation to any other columns.

More specifically, TUS [32] considers two tables to be unionable if they have multiple columns (*i.e.*, attributes) falling into similar domains. To find the unionable columns, it takes into account three factors: value overlap, ontology similarity, and natural language similarity, It first generates multiple LSH indexes to efficiently search for similar tables in the lake. Then, TUS traverses LSH-retrieved candidate tables, and uses the three factors to compute a union score for each candidate and query table. The performance bottleneck is in computing the natural language similarity because it involves a time-consuming matrix inversion operation. Thus, the online time complexity is $O(d^3\mathcal{S})$ (d is the dimension of embedding of a cell value), which is rather expensive for large data lakes.

Similarly, D3L [3] measures the similarity between two columns from 5 aspects: attribute name, attribute extent, word-embedding of attributes, format representation, and domain distribution. D3L utilizes LSH–based indexes to efficiently search similar columns in a data lake. The offline time (space) complexity of D3L is linear to the number of cell values (columns) in all the tables.

Contextual methods consider the contextual information of multiple columns within each table. Santos [24] uses a Knowledge Base (KB) to build a semantic graph for the columns of a table, where each node is a column, and the edge is the relationship deduced from the KB. When a query table comes, its graph will be compared with graphs (tables) in the data lake such that the column semantics are considered. The offline space complexity of Santos is $O(n^3m\mid\mathcal{T}\mid^2)$, which is extremely high and thus not scalable to large data lakes. (4) Starmie [17] uses pre-trained language models for union tables. It uses a lightweight contrastive learning method to train column encoders in an unsupervised way. The encoding of each column also considers other columns within a table, thus capturing the contextual semantics. It then encodes columns in the data lake as vectors and builds an HNSW index to accelerate the search.

Pros & Cons. In general, contextual methods are superior to column-independent methods, because to be unionable a pair of tables

should share the same semantics (*e.g.*, T_4 and T_5 in Figure 1 are about movies). This can be captured by the contextual methods.

4.3 Methods for Table Join and Union Search

In general, schema matching methods can support join/union search as they capture the similarity of schemas. We evaluate three typical methods. The first two use traditional similarity-based methods, while the third incorporates machine learning techniques.

Frt12 [36] introduces a framework to measure the relatedness of tables, which are then used for union or join search. For union, Frt12 uses KB to identify similar attributes and entities between the tables and computes a similarity score based on the overlap of these elements. For join, Frt12 determines whether two tables have complementary schemas, that is, if they have attributes that can be combined through a join operation. Then Frt12 considers the probability of complementary attributes appearing simultaneously in the same table. In the online process, Frt12 has to examine all tables in $\mathcal{T}$, which is rather expensive on large data lakes.

InfoGather [41] uses a large corpus of HTML tables to augment a given table, which can be regarded as table union or join search. The key idea is to first organize tables in the data lake as a graph, where nodes denote columns and edges correspond to potential matches. Then given a query table, InfoGather finds tables that can be unioned or joined using direct or indirect matching among tables in the graph. The time complexity of the offline process is high. The reason is that to build the graph, each node (column) should consider its similarity with others. The number of columns that a node should consider largely depends on the parameter $\mathcal{K}$, and thus the complexity is dominated by $\mathcal{KN}$. In practice, $\mathcal{K}$ is always large to achieve acceptable accuracy, making InfoGather rather expensive on large data lakes.

Aurum [18] supports both join and union search. First, the schema of each column is encoded using word embeddings. Then, all columns are organized as a graph using the LSH index, where each node corresponds to a column and each edge connects two nodes if they have similar embeddings. When a query table comes, Aurum encodes all columns in the table into embeddings and finds similar columns in the data lake with locality-sensitive hashing (LSH). The nearby tables in the graph are also retrieved.

Pros & Cons. Overall, traditional schema matching methods do not show superior effectiveness and efficiency because (1) they cannot well capture semantic information and (2) their matching is expensive, as unlike vector similarity search they do not have a good index to use. Although Aurum uses embeddings to capture semantics, it has limitations, because it only considers the schema, while ignoring the cell values.

5 EVALUATION

In this section, we evaluate existing table discovery approaches on LakeBench benchmark, and analyze the results. Our evaluation aims to answer the following questions.

- **Q1**: How do different algorithms perform on the benchmark in effectiveness?
- **Q2**: How efficient are these algorithms, especially on large data lakes?

- **Q3**: What about the memory consumption of these algorithms, especially on large data lakes?
- **Q4**: How do these algorithms perform on different categories of queries with different properties?
- **Q5**: Do these algorithms perform consistently across synthetic and real queries?
- **Q6**: How do table pre-training based methods perform on table discovery?

5.1 Setup

Metric. Following existing works [3, 24, 32], we use Precision@k ($P@k$) and Recall@k ($R@k$) to measure the effectiveness. Formally, given a query table T_q, we use $\mathcal{T}_q$ to denote the set of k tables retrieved by a specific method, and use $\mathcal{T}_g$ to denote the set of ground truth unionable/joinable tables. Then, the $P@k$ and $R@k$ of T_q are defined as $P@k = \frac{\mathcal{T}_g \cap \mathcal{T}_q}{T_q}$, $R@k = \frac{\mathcal{T}_g \cap \mathcal{T}_q}{T_g}$. For each method, we report its average $P@k$ and $R@k$ of all queries.

Environments. We implement all experiments in Python and run experiments on an Ubuntu Server with four Intel(R) Xeon(R) Gold 6148 2.40GHz CPUs having in total of 80 cores, two Nvidia Geforce 4090 GPUs, 1TB DDR4 main memory, and 6TB SSD. The same environments make sure a fair comparison over different methods.

Settings. For the top-k based table discovery, we set $k = 20$ for WebTable and WebTable Large and $k = 50$ for OpenData and OpenData Large. For candidate generation, we set K as 100. For the HNSW index, we set the number of neighbors of each node as 30. For LSH Ensemble, we set the containment threshold as 0.7, and the number of partitions (hash functions) as 8 (256). For Pexeso, we set the distance threshold, the column joinability threshold, the number of pivot vectors and the number of levels in a hierarchical grid as 0.3, 0.4, 3 and 4 respectively. The dimension of embedding generated by Starmie and DeepJoin are both 768. For Aurum, TUS and LSH Ensemble, we utilize a minhash dimension of 128. For D3L, Starmie, Santos and LSH Ensemble, we largely leverage their open-sourced implementation, and for others, we implement them based on the ideas of the original paper because of the lack of codes or using different programming languages.

5.2 Overall Efficacy Comparison

Effectiveness (Q1). We measure the $P@k$ and $R@k$ of both union and join search.

Union Search. Figure 4 and 5 report the results of $P@k$, $R@k$ on the four data lakes. Overall, traditional schema matching-based methods (*e.g.*, InfoGather and Frt12) perform the worst because they rely on the string similarity to compute the unionability among columns without considering the semantics. Aurum, D3L, and TUS outperform them because they all use word embeddings to capture the semantics. D3L performs better because it considers more factors (*e.g.*, attribute extent, format representation) than TUS and Aurum to measure the unionability.

Contextual methods (*e.g.*, Santos and Starmie) outperform all the above methods because they take contextual information among columns into consideration. Starmie performs the best because it fine-tunes the pre-trained language model that produces better embeddings and more effectively captures the contextual information.
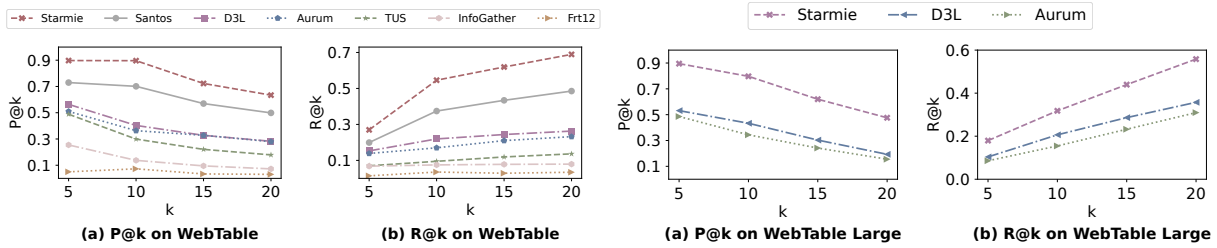
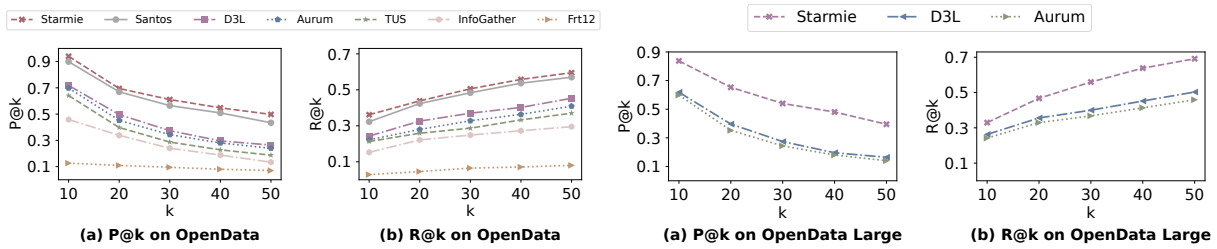**Figure 4: Effectiveness on WebTable / WebTable Large for Union Queries.**

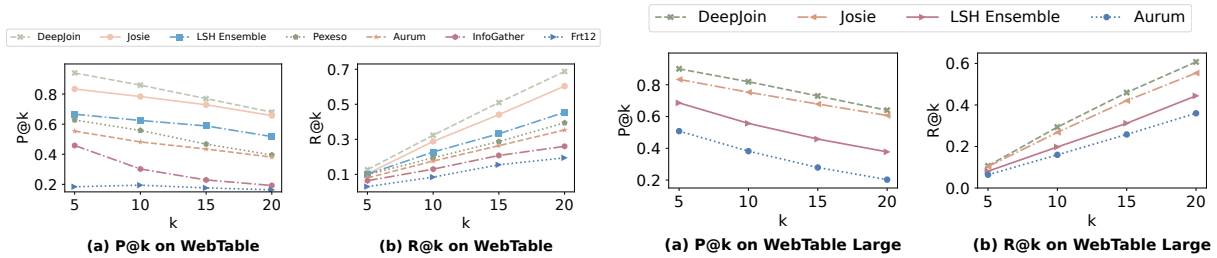**Figure 5: Effectiveness on OpenData / OpenData Large for Union Queries.**

Figure 6: Effectiveness on WebTable / WebTable Large for Join Queries.
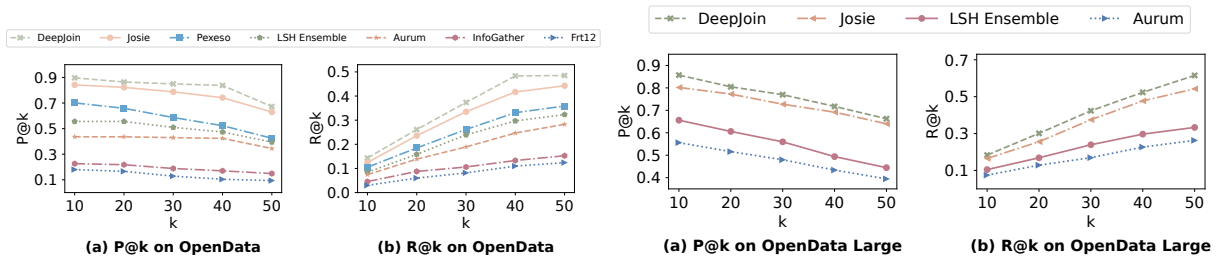

Figure 7: Effectiveness on OpenData / OpenData Large for Join Queries.

Overall, on all datasets, a larger k leads to a lower $P@k$ but higher $R@k$, because returning more tables has a larger chance to find more unionable tables, but may introduce more false positives. Besides, even for the state-of-the-art methods (*e.g.*, Starmie, DeepJoin), we can observe that $P@k$ always drops to a relatively low value at the end. The reason is that for some queries, especially the real queries, the number of ground truth tables is small, so the precision will be low if k is large (details will be discussed in **Q5**).

[Opportunity:] Choosing an appropriate k is very challenging because it is highly related to the characteristics of data lakes and queries. However, an appropriate k is rather critical because it offers a good trade-off between precision and recall while saving the users' efforts in examining the returned results.

Join Search. Figure 6 and Figure 7 report the effectiveness of the join search methods.

With reasons similar to union search, schema matching-based methods do not perform well. Pexeso performs better because it converts columns into embeddings, considering the semantic information. Set domain methods (*e.g.*, Josie and LSH Ensemble) outperform the above method because they identify the exact overlaps among columns, while in reality, a large proportion of joinable columns have exact overlaps. Josie performs better than LSH Ensemble, because Josie can identify exact top-k overlaps while LSH Ensemble focuses on estimating the overlaps. Overall, DeepJoin performs the best because it leverages the pre-trained language model to capture the semantics better. In this way, besides the exact overlap, it captures fuzzy overlaps like (Apple, Apple Inc.) as well, thus outperforming Josie.

Summary I: For union search, semantic information, especially the contextual information across multiple columns, matters a lot. For join search, column overlap is a key factor to consider, while in general a well-performing semantic-aware approach can capture the overlaps as well.

Efficiency (Q2). We measure the time of both offline index building and online query processing.

Offline. For union search, we observe from Table 6 that on WebTable, by the offline indexing time, we can see that the HNSW index is more efficient to build than others. Among these methods, InfoGather is the slowest (15 days) because it has to consider a large number of relationships between column pairs, leading to a high time complexity. TUS are slow because they consider all the cell values in the data lake. We observe a similar ranking on OpenData. The key difference is Santos becomes the slowest because the largest number of columns/rows in a table on OpenData becomes much larger. Also, the HNSW-based index becomes faster because there are fewer columns in OpenData. Note we do not report the result of InfoGather on large data lakes, because it takes too long to finish. In addition to the indexing time, it takes the natural language domain methods extra time to generate the embeddings. For example, using our two GPUs, this takes Starmie 10 hours.

For join search (Table 5), methods that construct the HNSW index are still the most efficient, while InfoGather is still the slowest. LSH Ensemble is not efficient enough because it needs to build multiple LSH indexes. The performance on OpenData is similar.

Online. We can observe from Table 6 that on WebTable, by the online efficiency, methods with the HNSW index are still the fastest because HNSW uses a hierarchical graph structure to organize the column embeddings. TUS is the slowest because it has a high complexity of calculating natural language similarity, which involves extremely time-consuming matrix inversion. Frt12 is not efficient because it has to iterate every table of the data lake. We set a maximum time duration of 3 days and exclude the results of Frt12 and TUS due to the fact that the online phase of these methods cannot be completed within this timeframe. For *join search* on WebTable, methods with the HNSW index are still the most efficient. Pexeso is the slowest because it has a high time complexity $O(\log \mathcal{A} \log \mathcal{R})$, where $\mathcal{R}$ is the number of distinct cell values, and thus we do not report the result on large data lakes. On OpenData, we have a similar observation.

Summary II: For efficiency, indexing matters the most in both online and offline processes. Embedding-based methods show superiority over other methods, because they not only consider semantics, but also benefit from many well-designed ANN indexes. For online efficiency, the HNSW-based index is no doubt the most efficient because of its hierarchical graph structure. For offline, building the HNSW-based index takes several hours, which is acceptable. Other methods (*e.g.*, LSH Ensemble) are more time-consuming, as they use more optimizations to improve the effectiveness.

Memory Usage (Q3). We also report the memory usage in Table 5 and 6. We can observe that for union search methods, on WebTable and Opendata, Santos consumes the largest memory because of its high space complexity ($O(n^3 m \mid \mathcal{T} \mid^2)$), where $n(m)$ is the largest number of columns (rows) in a table from $\mathcal{T}$ and $\mid \mathcal{T} \mid$ is the number of tables. It thus fails on the large data lakes. Next, D3L constructs multiple LSH indexes to compute the column similarity, leading to high memory usage on WebTable. However, on OpenData, the memory usage is smaller because of the much smaller number of columns in this data lake. The memory usage of Starmie is also relatively large on WebTable because each node in the graph index corresponds to a column embedding, and we have to load the entire graph into the memory.

For join search methods, on WebTable, LSH Ensemble is the most memory-thirsty because of using multiple LSH indexes. DeepJoin and Starmie have relatively large memory usage on WebTable because of the large HNSW index, while on OpenData, their memory usage is small because of the small number of columns. On the other hand, the space complexity of Pexeso is highly correlated to the number of distinct values, so it consumes more memory on OpenData than on WebTable.

Summary III: HNSW-based index is not memory efficient, because it has to load the entire graph into the memory, where nodes correspond to the embeddings of all columns, and the edges represent the relationships between the columns. Therefore, to fully explore its high efficiency and effectiveness, users need to configure a high-memory server.

[Opportunity:] In terms of effectiveness and computation efficiency, the HNSW-based index is preferable although it is not very memory efficient. In case users do not have a high memory machine, product quantization [22] is a good alternative, as it is memory efficient and typically shows better accuracy than LSH.

5.3 Different Query Categories (Q4)

We categorize a number of query-candidate pairs with ground truth (whether they are joinable/unionable) based on different factors that make them unionable/joinable or not. In general, these factors represent that how much semantic information should be considered when determining the joinability/unionability of one pair. Each category includes 100 pairs. Then, we apply different types of algorithms over each category of queries, aiming to answer the following question: *whether there exists an algorithm that performs well over all categories.*

Table 5: Efficiency and Memory Usage of Table Join Search.

Methods	WebTable				WebTable Large				OpenData				OpenData Large			
	Offline		Online		Offline		Online		Offline		Online		Offline		Online	
	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem
Josie	3h	28G	0.9s	44.5G	17h	172G	3.29s	262G	13h	85G	0.34s	11.3G	3.4d	508G	1.16s	73G
LSH Ensemble	3.7h	268G	1.53s	289G	21.6h	851G	4.7s	872G	2.1h	4.9G	0.43s	6.7G	10h	28.3G	0.59s	32G
Pexeso	2.7h	33.9G	62s	49.5G	–	–	–	–	2.6h	114G	56.8s	130G	–	–	–	–
DeepJoin	16m	138G	0.37s	96G	1.9h	712G	0.49s	410G	6s	1.9G	0.13s	1.6G	48s	12.1G	0.29s	8.9G
Frt12	2.2h	2.3G	23.5s	0.4G	–	–	–	–	0.5h	4.3G	9.6s	0.36G	–	–	–	–
InfoGather	15d	59.2G	0.73s	44G	–	–	–	–	22.7h	128.3G	0.22s	156.2G	–	–	–	–
Aurum	12.5m	40G	0.15s	30.4G	0.5h	210G	0.2s	157G	9s	0.6G	0.09s	0.45G	1.3m	3.9G	0.45s	3.2G

Table 6: Efficiency and Memory Usage of Table Union Search.

Methods	WebTable				WebTable Large				OpenData				OpenData Large			
	Offline		Online		Offline		Online		Offline		Online		Offline		Online	
	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem
TUS	9.6h	57G	40.6s	142G	–	–	–	–	3.3d	45G	50.2s	127G	–	–	–	–
D3L	16h	300G	9.2s	280G	4.9d	600G	24.2s	350G	4.6d	55G	3.4s	55G	17.9d	94G	3.8s	116G
Starmie	12.5m	141G	0.3s	98G	2.1h	780G	0.33s	499G	7s	2.2G	0.1s	1.8G	1.1m	13.2G	0.15s	9.6G
Santos	1.1d	320G	2.5s	130G	–	–	–	–	6.3d	809G	40.7s	374G	–	–	–	–
Frt12	2.1h	2.1G	23.2s	0.2G	–	–	–	–	0.9h	3.6G	9.7s	0.3G	–	–	–	–
InfoGather	15d	59.2G	6.5s	46.3G	–	–	–	–	22.7h	128.3G	5.11s	174.2G	–	–	–	–
Aurum	12.5m	40G	0.16s	32.4G	1.5h	210G	0.24s	159G	9s	0.6G	0.1s	0.6G	1.3m	3.9G	0.49s	3.5G

Categories of join pairs. We construct three categories of query-candidate pairs. Next, we use the examples in Figure 1 of Section 1 to introduce how these pairs are picked.

[C_1^J: Exact overlap.] In this category, the specific column (*e.g.*, NVIDIA, Tesla, Apple) of the query table has exact overlaps with a column (*e.g.*, Apple, Tesla, Google) of the candidate table. The pairs in this category are all joinable (positive pairs).

[C_2^J: Fuzzy overlap.] This category is more complicated than C_1^J, because the two to-be-joined columns share many fuzzy overlapping values. For example, in Figure 1, the first column of T_1 (Corp: NVIDIA, Tesla, Apple) has fuzzy overlaps with the first column of T_2 (Corp: NVIDIA Corp., Apple Inc., Amazon). The pairs in this category are all joinable (positive pairs).

[C_3^J: Semantically similar but no overlap.] This is a *hard negative* category in which each pair is not joinable. Using the first column of T_1 (Corp: NVIDIA, Tesla, Apple) and the first column of T_3 (Corp: Alibaba, Tencent, Baidu) as an example, although they are semantically similar, they are not joinable due to the lack of (fuzzy) overlaps. However, many semantic-aware methods might mis-classify them as joinable.

Categories of union pairs. We ask the experts to pick two categories of pairs from their labeling results.

[C_1^U: Unionable but low column relevance scores.] As shown in Figure 1, although only two pairs of columns can be unioned, T_4 is still unionable with T_5 because they have similar table-level semantics. We regard this as a hard positive case as some methods that consider column-wise similarity independently will give low column relevance scores, produce a low table-level score, and predict these pairs as not unionable.

[C_2^U: High column relevance scores but not unionable.] In Figure 1, although 3 out of 4 columns in T_6 can be unioned with T_4, they are not unionable at table level because of the different semantics. This

category can be regarded as a hard negative case, because some methods will give such pairs high column relevance scores, leading to high table-level scores.

Metric. We use *accuracy* to measure the effectiveness of an algorithm over a query category that contains only positive (joinable/unionable) or negative (not joinable/unionable) pairs. Accuracy is determined by the ratio of correctly classified pairs to the total number of pairs. For a set of positive pairs, $C_i^J = (T, T')$, correct classification occurs when the top-k results for query T include T'. Conversely, in a query category with exclusively negative pairs, the classification is considered to be correct when the top-k results for query T *do not* contain T'.

Observations for join pairs: We use DeepJoin, Josie and Aurum to test the three join categories, and the results are shown in Table 7. DeepJoin is a typical algorithm that considers column semantics, while Josie uses exact overlaps between columns. Aurum represents the schema matching-based methods.

We can observe that for C_1^J, DeepJoin and Josie perform competitively. The reason is that Josie performs well when there are many exact overlaps between two columns, as it can precisely identify exact overlaps. DeepJoin is comparable because significant overlaps always indicate high semantic similarity, which can be well captured by DeepJoin. However, Josie may not always achieve the best performance in this case with exact overlaps. For instance, given a specific query column (*e.g.*, Last Name: Washington, Lincoln, Henry) which represents the last names of the humans, it exactly overlaps with a candidate column (*e.g.*, Avenue: Lincoln, Washington Ave., Madison Rd.) which instead represents street names. As they are in different domains, the two columns are not joinable despite they are overlapping. However, Josie may incorrectly identify this as one of the top-k results because of the overlaps, but DeepJoin can handle this case because it well considers the semantics.

Table 7: Accuracy of Different Join Query Categories.

Join Methods	Top-k	C_1^J	C_2^J	C_3^J
DeepJoin	$k = 10$	51%	33%	78%
	$k = 20$	81%	55.5%	63%
Josie	$k = 10$	50%	0%	100%
	$k = 20$	81.5%	0%	100%
Aurum	$k = 10$	37%	8%	44%
	$k = 20$	56%	13.5%	30%

Table 8: Accuracy of Different Union Query Categories.

Union Methods	Top-k	C_1^U	C_2^U
Starmie	$k = 10$	34%	81%
	$k = 20$	77.5%	76%
D3L	$k = 10$	19.5%	47%
	$k = 20$	46%	34%
Aurum	$k = 10$	16%	43.5%
	$k = 20$	41%	29%

For C_2^J, DeepJoin performs the best, as expected. This is because DeepJoin explicitly considers the semantics, thus well capturing the fuzzy overlaps between two columns. Josie does not perform well because it only takes exact overlaps into account and cannot capture the fuzzy overlaps. Aurum has poor performance because it only captures the schema information.

For C_3^J, Josie performs the best. For example, when $k = 20$, its accuracy is 100%, which indicates that none of these query-candidate pairs in C_3 belong to the top-20 of the queries. DeepJoin focuses more on the column semantics, and thus many semantically similar pairs without overlaps will be incorrectly identified as joinable, leading to a lower performance. Aurum does not perform well because it cannot precisely capture the overlaps.

Summary IV: No method is always effective in all scenarios. The main reason is that to be joinable, two columns should be semantically similar but also have (fuzzy) overlaps. Although a well-designed semantic-aware method in general works well, it may fail when two columns share similar semantics but no overlaps.

[**Opportunity:**] A potential research direction is to design a strategy that effectively distinguishes the fuzzy overlaps from semantically similar but non-overlapping columns. Combining it with exact overlaps might lead to a more accurate join search algorithm. *Observations for union pairs:* We use Starmie and D3L to test the two categories of queries as the representatives of two types of methods: considering contextual semantics or column-independent semantics. We also use Aurum to represent schema matching based methods. The results are shown in Table 8. We observe that for C_1^U, Starmie performs the best. For example, when setting $k = 20$, Starmie has an accuracy of 77.5%, outperforming D3L (46%). The reason is that it fine-tunes a pre-trained language model, which takes into account the contextual information among columns. D3L does not perform well because it only considers each pair of columns of two tables independently. For C_2^U, Starmie outperforms D3L. The reason is that D3L computes the similarity between each column pair individually. If many column pairs are unionable, its table-level relevance score will be high. But the two tables might not be unioned if they are in fact not in a similar context. Schema matching based method does not perform well because it does not well consider the semantics.

Summary V: The method (*i.e.*, Starmie) considering contextual information performs better on union search. The reason is that different from join search, semantic information matters more than the overlaps on union search. When an expert labels if two tables are unionable, the similarity on the table-level semantics is the most critical factor, which is well captured by contextual modeling.

5.4 Real Queries V.S. Synthetic Queries (Q5)

For real queries, only a few tables in the lake can be unioned or joined with them. For example, on WebTable and OpenData, 80% real queries have no more than 10 joinable/unionable queries. But for synthetic tables, on average, each query has 30 unionable tables on OpenData. We also observe that almost all algorithms perform better on synthetic queries than on real queries. Because for synthetic queries, the similarities between queries and the candidate tables are relatively high, which makes it easier for algorithms to identify these candidate tables.

Besides, we observe that different algorithms perform consistently across real and synthetic queries. Taking union as an example (Figure 9), methods that incorporate contextual semantics, such as Starmie and Santos, outperform those focusing on independent column semantics (e.g. Aurum, TUS) as well as schema matching-based approaches (e.g. InfoGather, Frt12) on both types of queries.

Summary VI: The real queries typically have much fewer joinable/unionable tables than the synthetic queries, thus harder to discover from a big data lake due to this sparsity. Therefore, the search algorithms tend to have a lower accuracy on these real queries. However, the rankings of the algorithms are consistent on real and synthetic queries.

5.5 Table Pre-training Methods (Q6)

We evaluate different table pre-trained models on union search, including TABERT [42], TABBIE [21]. Similar to [29], we use them to generate column embeddings, on which we compute the unionbility score and discover top-k tables. For TABBIE and TABERT, we also use their open-sourced implementation. TABERT is a pre-trained language model that simultaneously captures the representations of structured tables and natural language phrases. TABBIE introduces a simple pre-training objective, namely identifying corrupt cells from tabular data. Then, we compare them with Starmie, D3L and InfoGather. In Figure 10, Starmie still performs the best, because based on BERT [12], it further leverages the contrastive training technique to fine-tune the model to better fit the table discovery
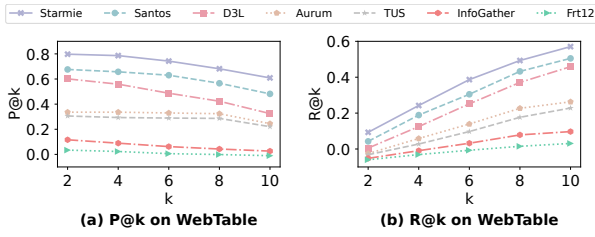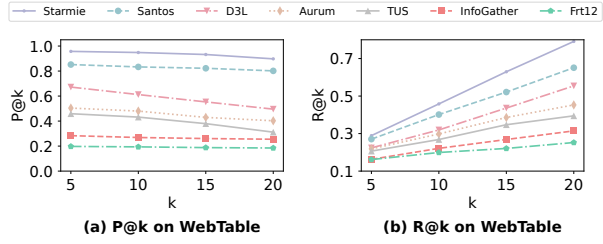
Figure 8: Effectiveness of Real Union Queries.

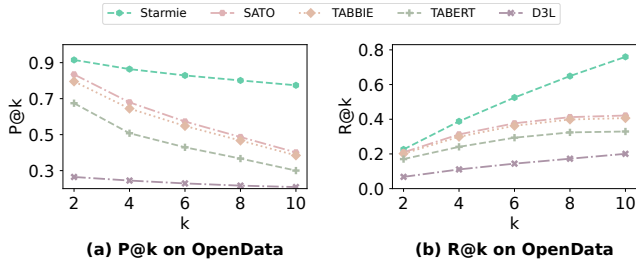Figure 9: Effectiveness of Synthetic Union Queries.

Figure 10: Effectiveness of Table Pre-training Methods.

task. However, these pre-training based models outperform other methods, because they can well capture the semantic information.

> Summary VII: In general, pre-training based methods achieve good effectiveness because they are able to produce more informative column representations. Fine-tuning the pre-trained model (*i.e.*, Starmie) can further improve the performance.

[Opportunity:] Although Starmie, which fine-tunes a pre-trained BERT model for the data discovery tasks, in general performs the best, its effectiveness (*R@k* is always smaller than 80%) still has a relatively large space to improve. Therefore, designing a better fine-tuning strategy or fine-tuning the larger language models like LLAMA2 [38] is a promising direction to explore.

6 RELATED WORK

Existing released datasets & queries. TUS [32] is the first one to define the table union search problem and propose an LSH-based method to solve it. It releases the datasets and queries used in experiments, which contain 5,000 tables collected from OpenData. They construct 1,000 queries, all of which are *synthetic queries* obtained through splitting large tables. Santos [24] improves the table union search strategy by considering the relationships between the columns. To evaluate their method, it labels *80* queries based on the column-to-column relations – a relatively small number of queries. The scale of its dataset is also limited – containing only 10,100 tables. Josie [45] discovers joinable tables by searching column pairs with exact overlaps. Josie uses two data lakes for evaluation. One is the same with TUS, and the other is sampled from WebTable. It picks 1,000 query columns from each data lake to evaluate the efficiency of its similarity search algorithm. Hence its queries are not very useful in evaluating the effectiveness of other methods

that take semantics into consideration. Valentine [25] evaluates how schema matching techniques perform in table discovery tasks. The evaluation is to match pairs of tables within existing datasets. Valentine creates 540 synthetic queries, which only support the evaluation of schema matching approaches. Different from above works, LakeBench builds a benchmark that offers multiple large-scale data lakes and diverse real & synthetic queries to thoroughly evaluate both table join and union search methods. In addition, an unpublished work [33] leverages the large language model (LLM) to generate data and queries for *table union search*. However, it produces a relatively small dataset with only 1,050 tables and covers a limited number of search methods.

Table discovery for ML. To address the data scarcity problem, recently, researchers have studied to acquire data from external resources (*e.g.*, data lake) to enrich the training set, to improve the model performance. At a high level, for tabular data, existing methods can be categorized into enriching features [8, 26, 28] and tuples [7, 27, 43], which can respectively leverage the table join and union search as a pre-processing step.

Keyword-based search. In the realm of keyword-based search for table discovery in data lakes, [4] introduces Octopus, a system that uses keyword-based search techniques to integrate structured data from the web. It describes several algorithms and operators that automate the data integration process, including the Search operator which takes a keyword query as input and returns a ranked list of table clusters. [34] presents a structured search engine that utilizes the vast amount of tables available on the web to provide multi-column table results in response to keyword queries.

7 CONCLUSION

In this paper, we construct a comprehensive benchmark for joinable and unionable table discovery. We build 4 data lakes and construct both synthesized and real queries. Additionally, we make great efforts to label the ground truth of thousands of queries. Finally, we sufficiently compare the effectiveness and efficiency of multiple table discovery algorithms and provide meaningful insights.

REFERENCES

[1] [n.d.]. OpenData. https://open.canada.ca/.

[2] [n.d.]. WebTable. https://webdatacommons.org/webtables/.

[3] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In 36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020. IEEE, 709–720.

[4] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. 2009. Data Integration for the Relational Web. Proc. VLDB Endow. 2, 1 (2009), 1090–1101. https://doi.org/10.14778/1687627.1687750

[5] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-loop Outlier Detection. In Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 19–33. https://doi.org/10.1145/3318464.3389772

[6] Chengliang Chai, Jiabin Liu, Nan Tang, Ju Fan, Dongjing Miao, Jiayi Wang, Yuyu Luo, and Guoliang Li. 2023. GoodCore: Data-effective and Data-efficient Machine Learning through Coreset Selection over Incomplete Data. Proc. ACM Manag. Data 1, 2 (2023), 157:1–157:27. https://doi.org/10.1145/3589302

[7] Chengliang Chai, Jiabin Liu, Nan Tang, Guoliang Li, and Yuyu Luo. 2022. Selective Data Acquisition in the Wild for Model Charging. Proc. VLDB Endow. 15, 7 (2022), 1466–1478. https://www.vldb.org/pvldb/vol15/p1466-li.pdf

[8] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David R. Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. Proc. VLDB Endow. 13, 9 (2020), 1373–1387. https://doi.org/10.14778/3397230.3397235

[9] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System. In 8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings. www.cidrdb.org. http://cidrdb.org/cidr2017/papers/p44-deng-cidr17.pdf

[10] Yuhao Deng, Chengliang Chai, Lei Cao, Nan Tang, Jiayi Wang, Ju Fan, Ye Yuan, and Guoren Wang. 2024. MisDetect: Iterative Mislabel Detection using Early Loss. Proc. VLDB Endow. 17, 6 (2024), 1159–1172. https://www.vldb.org/pvldb/vol17/p1159-chai.pdf

[11] Yuhao Deng, Qiyan Deng, Chengliang Chai, Lei Cao, Nan Tang, Ju Fan, Jiayi Wang, Ye Yuan, and Guoren Wang. 2024. IDE: A System for Iterative Mislabel Detection. In Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago, Chile, June 9-15, 2024. ACM.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[13] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. In 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021. IEEE, 456–467.

[14] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2022. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. CoRR abs/2212.07588 (2022).

[15] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2022. MATE: Multi-Attribute Table Extraction. Proc. VLDB Endow. 15, 8 (2022), 1684–1696. https://doi.org/10.14778/3529337.3529353

[16] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. 2023. Table Discovery in Data Lakes: State-of-the-art and Future Directions. In Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023, Sudipto Das, Ippokratis Pandis, K. Selçuk Candan, and Sihem Amer-Yahia (Eds.). ACM, 69–75. https://doi.org/10.1145/3555041.3589409

[17] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. Proc. VLDB Endow. 16, 7 (2023), 1726–1739.

[18] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018. IEEE Computer Society, 1001–1012.

[19] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing Google's Datasets. In Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016. ACM, 795–806. https://doi.org/10.1145/2882903.2903730

[20] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Managing Google's data lake: an overview of the Goods system. IEEE Data Eng. Bull. 39, 3 (2016), 5–14. http://sites.computer.org/debull/A16sept/p5.pdf

[21] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 3446–3456. https://doi.org/10.18653/V1/2021.NAACL-MAIN.270

[22] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. IEEE Trans. Pattern Anal. Mach. Intell. 33, 1 (2011), 117–128. https://doi.org/10.1109/TPAMI.2010.57

[23] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomás Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers, Mirella Lapata, Phil Blunsom, and Alexander Koller (Eds.). Association for Computational Linguistics, 427–431. https://doi.org/10.18653/V1/E17-2068

[24] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2022. SANTOS: Relationship-based Semantic Table Union Search. CoRR abs/2209.13589 (2022).

[25] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021. IEEE, 468–479. https://doi.org/10.1109/ICDE51399.2021.00047

[26] Arun Kumar, Jeffrey F. Naughton, Jignesh M. Patel, and Xiaojin Zhu. 2016. To Join or Not to Join?: Thinking Twice about Joins before Feature Selection. In Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016. ACM, 19–34. https://doi.org/10.1145/2882903.2882952

[27] Yifan Li, Xiaohui Yu, and Nick Koudas. 2021. Data Acquisition for Improving Machine Learning Models. Proc. VLDB Endow. 14, 10 (2021), 1832–1844. https://doi.org/10.14778/3467861.3467872

[28] Jiabin Liu, Chengliang Chai, Yuyu Luo, Yin Lou, Jianhua Feng, and Nan Tang. 2022. Feature Augmentation with Reinforcement Learning. In 38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022. IEEE, 3360–3372. https://doi.org/10.1109/ICDE53745.2022.00317

[29] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Trans. Pattern Anal. Mach. Intell. 42, 4 (2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[30] Renée J. Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q. Pu, and Periklis Andritsos. 2018. Making Open Data Transparent: Data Discovery on Open Data. IEEE Data Eng. Bull. 41, 2 (2018), 59–70. http://sites.computer.org/debull/A18june/p59.pdf

[31] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. Proc. VLDB Endow. 12, 12 (2019), 1986–1989. https://doi.org/10.14778/3352063.3352116

[32] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. Proc. VLDB Endow. 11, 7 (2018), 813–825.

[33] Koyena Pal, Aamod Khatiwada, Roee Shraga, and Renée J. Miller. 2023. Generative Benchmark Creation for Table Union Search. CoRR abs/2308.03883 (2023). https://doi.org/10.48550/arXiv.2308.03883 arXiv:2308.03883

[34] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web using Column Keywords. Proc. VLDB Endow. 5, 10 (2012), 908–919. https://doi.org/10.14778/2336664.2336665

[35] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. CoRR abs/1910.01108 (2019). arXiv:1910.01108 http://arxiv.org/abs/1910.01108

[36] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 817–828.

[37] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. MPNet: Masked and Permuted Pre-training for Language Understanding. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/c3a690be93aa602ee2dc0ccab5b7b67e-Abstract.html

[38] Hugo Touvron, Louis Martin, and Kevin Stone et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. CoRR abs/2307.09288 (2023). https://doi.org/10.48550/ARXIV.2307.09288 arXiv:2307.09288

[39] Jiayi Wang, Chengliang Chai, Nan Tang, Jiabin Liu, and Guoliang Li. 2022. Coresets over Multiple Tables for Feature-rich and Data-efficient Machine Learning. Proc. VLDB Endow. 16, 1 (2022), 64–76. https://doi.org/10.14778/3561261.3561267

[40] Pei Wang, Ryan Shea, Jiannan Wang, and Eugene Wu. 2019. Progressive Deep Web Crawling Through Keyword Queries For Data Enrichment. In Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019. ACM, 229–246. https://doi.org/10.1145/3299869.3319899

[41] 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 ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 97–108.

[42] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics,

8413–8426. https://doi.org/10.18653/V1/2020.ACL-MAIN.745

[43] Jinsung Yoon, Sercan Ömer Arik, and Tomas Pfister. 2020. Data Valuation using Reinforcement Learning. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research), Vol. 119. PMLR, 10842–10851. http://proceedings.mlr.press/v119/yoon20a.html

[44] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. Proc. VLDB Endow. 13, 11 (2020), 1835–1848. http://www.vldb.org/pvldb/vol13/p1835-zhang.pdf

[45] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 847–864.

[46] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. Proc. VLDB Endow. 9, 12 (2016), 1185–1196.