

# Diverse Unionable Tuple Search: Novelty-Driven Discovery in Data Lakes

Aamod Khatiwada  
Northeastern University  
Boston, Massachusetts, USA  
khatiwadaamod@gmail.com

Roe Shraga  
Worcester Polytechnic Institute  
Worcester, Massachusetts, USA  
rshraga@wpi.edu

Renée J. Miller  
U. Waterloo & Northeastern U.  
Waterloo, Ontario, Canada  
rjmiller@uwaterloo.ca

## Abstract

Unionable table search techniques input a query table from a user and search for data lake tables that can contribute additional rows to the query table. The definition of unionability is generally based on similarity measures which may include similarity between columns (e.g., value overlap or semantic similarity of the values in the columns) or tables (e.g., similarity of table embeddings). Due to this and the large redundancy in many data lakes (which can contain many copies and versions of the same table), the **most** unionable tables may be identical or nearly identical to the query table and may contain little new information. Hence, we introduce the problem of identifying unionable tuples from a data lake that are diverse with respect to the tuples already present in a query table. We perform an extensive experimental analysis of well-known diversity algorithms applied to this novel problem and identify a gap that we address with a novel, clustering-based tuple diversity algorithm called DUST. DUST uses a novel embedding model to represent unionable tuples that outperforms other tuple representation models by at least 15% when representing unionable tuples. Using real data lake benchmarks, we show that our diversification algorithm is more than six times faster than the most efficient diversification baseline. We also show that it is more effective in diversifying unionable tuples than existing diversification algorithms.

## Keywords

Data Discovery, Data Integration, Data Lakes, Tuple Diversification

## 1 Introduction

Content-based table search (or table as a query) has become the dominant paradigm for table discovery in data lakes [8, 10, 18, 20], particularly when metadata may be missing, ambiguous, inconsistent, or incomplete [27, 36]. In this paradigm, users input a table as a query, and the table search system uses the data available within the query table to retrieve relevant tables. Among different types of relevance search [10, 18, 25], we focus on *Table Union Search* [37], where the objective is to search for additional "unionable" tables. Note that an important end goal of searching for the unionable tables is to add new rows to a given query table. However, table union search techniques return the data lake tables that are most unionable to the query table [2, 24, 37]. They measure unionability between tables based on different similarity measures including value overlap between the columns [2, 37], knowledge graphs concept similarity [24, 37], word embedding similarity [37], similarity of table representations [11, 20], and so on. But such similarity-based techniques tend to return tables containing similar or even redundant tuples (with respect to the

query table) in the top rankings. In fact, a table is most similar to itself.

**EXAMPLE 1.** Consider Query Table (a) and Data Lake Tables (b), (c) and (d) in Fig. 1. Tables (a), (b) and (d) are about parks and their different properties, and both Tables (b) and (d) are unionable with the query table. Table (c) is about paintings and as it shares only the country attribute with (a), it is determined to be not unionable with the query table. Since table union search techniques [11, 20, 24, 37] measure similarity between query and data lake tables to infer unionability, they return Table (b) as the most unionable table (Table (d) while unionable is less similar to the query). However, Table (b) is mostly a copy of the query table with a single new tuple and thus does not extend the user's analysis much (we illustrate this in Table (e) which is "most unionable"). In contrast, Table (d) is both unionable and contains tuples with new information.

If the data lake contains tables with overlapping tuples, these are likely to be returned by a union search method. Others have documented the tremendous redundancy in real data lakes [57], noting that almost 90% of the data found in newly created datasets duplicates information already present in existing datasets in a lake [45]. So, the top-ranked tables returned by a similarity-based union search technique may not offer much new information. An example of the impact of such redundancy could be if one were trying to integrate the discovered tables to achieve a fairness goal [35, 47, 53] where most approaches assume that the discovered tables have enough tuples available to satisfy fairness constraints [35]. However, if redundancy is high, this may not be the case. Table search is used to find training data for machine learning models [14, 19] and redundancy in the discovered data may hamper the model's ability to generalize. A user study on diversifying information retrieval (IR) search results reflects that when looking for new information, users are interested in finding more information within their specific subtopics of interest, rather than arbitrarily looking for any new information [52]. In our case, for instance, a user is looking for information on new parks that are not present in the query table. In this paper, we present a novel evaluation of the ability of existing diversity algorithm to diversify the tuples returned by unionability methods. We further offer DUST, a method designed to address gaps identified in existing algorithms and scalably provide a diverse set of unionable tuples. Referring to Example 1 (Fig. 1), our proposed approach will find unionable tuples from a set of unionable tables where the tuples add new and diverse information to the query table as illustrated by Table (f).

**Tuple vs. Table Diversity.** Our important design decision is to return a set of diverse, unionable *tuples* (which may come from different tables), rather than a set of diverse, unionable *tables*. Fig. 2 provides a motivation for this decision. Using any tuple or table embedding technique, we can plot (using PCA) a two-dimensional representation of the embedding similarity among

EDBT '26, Tampere (Finland)

© 2025 Copyright held by the owner/author(s). Published on OpenProceedings.org under ISBN 978-3-98318-102-5, series ISSN 2367-2005. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

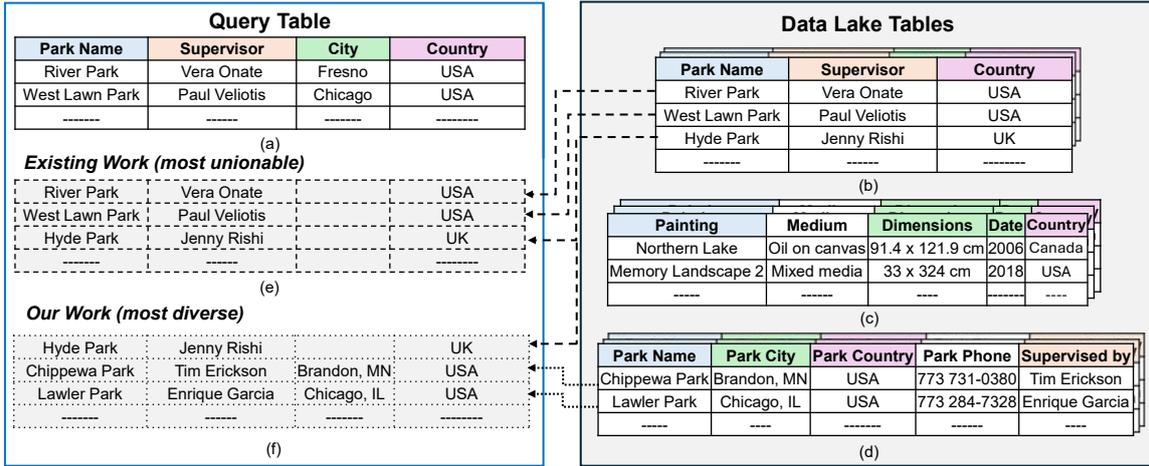


Figure 1: Table (a) is a Query Table (left) and Tables (b)-(d) are Data Lake Tables (right). Existing Work will add the tuples in Table (e) (most unionable) while our work will return the ones in Table (f), which are also the most diverse.

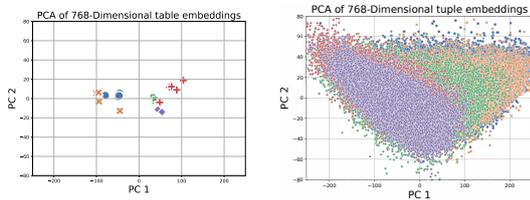


Figure 2: Table (left) and Tuple (right) embedding distributions of unionable tables (tuples) from Open Data [24]. Each color/marker represents a distinct set of unionable tuples/tables. Tables from different sets are non-unionable.

unionable and non-unionable tuples or tables. Fig. 2 does this using embeddings we will introduce in Sec. 4. On the left, we plot embeddings of five sets (each set in a different color/marker) of unionable tables from the SANTOS benchmark which contains tables from Open Data [24]. Tables from different sets are non-unionable. While there is some clustering structure in the plot, the unionable tables are not terribly diverse in the embedding space and even some non-unionable tables have quite similar representations. This shows that the diversity of the tables is limited. The right side contains the same plot for unionable and non-unionable tuples. Tuples are spread around the space much more, and even unionable tuples (with same color/marker) can appear very far apart in the embedding space. This suggests that diversifying unionable tables may have limited effect whereas selecting a diverse set of tuples from a set of unionable tables may well achieve our goal of providing new and diverse search results. In addition, directly searching for diverse unionable tuples may be infeasible since it requires an index over all tuples in a lake (rather than tables).

Extensive IR research, particularly in recommendation systems and web search, has also focused on the diversification of search results [1, 15, 50]. While IR solutions, at large, address very large repositories of data (e.g., documents), when it comes to diversification, they typically focus on smaller sets [5, 15, 50]. Diversification, in this case, is applied as a second step over human-consumable sets of information such as web pages or items on an e-commerce website. Accordingly, they experiment with relatively small sets (e.g., a prior work considers relevance item set size up to 5000 and diverse output result size up to 35 [51]; and a popularly used web search dataset has a set of

queries each with 100 relevant google web search results among which a small (typically, 5 or 10 items out of the 100) diverse set can be extracted [15]). In our scenario, the challenge lies in diversifying large sets (tens of thousands) of unionable tuples, which are intended to be processed in automated pipelines for downstream tasks [14].

Our method DUST (Diverse Unionable Tuple Search), which given a query table, searches for unionable tables from a data lake, and post-processes them to output k-diverse tuples that are diverse with respect to the query table and each other. We do not assume a small set of retrieved items and we do not assume a small k as in information retrieval, rather DUST considers outputting (a potentially large) set of diverse tuples from a larger set of unionable tuples. We further fine-tune a transformer-based model that represents the unionable tuples in an embedding space. We evaluate unionable table search techniques, emphasizing their effectiveness in achieving diversification. We further adopt and combine ideas from diversification and table union search to present a new algorithm. Specifically, our contributions are:

- **Diverse Unionable Tuple Search.** We identify and address the new problem of searching for diverse tuples from a data lake that can be unioned with the given query table.
- **Tuple Diversification Algorithm.** We present DUST, a new algorithm to diversify unionable data lake tuples with respect to the query table. The algorithm also uses a transformer-based fine-tuned model for tuple representation.
- **Extensive Empirical Evaluation.** We run experiments to show the effectiveness and efficiency of DUST in searching for diverse unionable tuples. Specifically, our novel DUST model outperforms baseline tuple embedding models by at least 15% in terms of accuracy, when used to represent unionable tuples.

Furthermore, we show that DUST’s tuple diversification algorithm is over six times faster than the most effective baseline [51]. Nevertheless, in most cases, our algorithm is more effective than all the baselines when diversifying tuples from existing table union search benchmarks [24, 39]. We further provide a case-study intuitively illustrating the practical benefits of the presented algorithm.

- **Open-source Code and Data.** We make all our resources open-source: <https://github.com/northeastern-datalab/dust>

## 2 Related Work

To the best of our knowledge, no prior work searches for a diverse set of unionable tuples from a data lake. So, we cover related work on finding unionable tables and on diversifying search results.

**Table Union Search.** Nargesian et al. [37] introduced the problem of searching for top-k unionable tables from a data lake, given a query table. They consider a data lake table to be unionable with a query table if they share unionable columns. To determine unionable columns, they defined three statistical tests based on value overlap, knowledge graph class overlap, and word embedding similarity between the columns. Zhu et al. [58] introduced the joinable table search problem with numerous extensions [8, 57] and Bogatu et al. [2] presented  $D^3L$  that searches for data lake tables that are related (meaning either unionable or joinable). Khatiwada et al. [24] introduced **SANTOS** that improves table union search accuracy by considering column semantics as well as the semantics of binary relationships between column pairs. In **Starmie**, Fan et al. [11] proposed a contrastive-learning framework that instead of looking at columns individually or looking at the binary relationships between the column pairs, captures the context of the entire table in the form of contextualized column embeddings and uses such embeddings to improve table union search accuracy. Hu et al. [20] proposed **AutoTUS** where they contextualized relationships between the column pairs to capture table contexts and use them to find unionable tables from the data lakes. Mirzaei and Rafiei [32] explore the search for similar data lake tables based on user preferences, incorporating diversity among their four major preference criteria. While their primary focus is on finding tables with new sets of columns, we concentrate on the addition of new tuples. Nevertheless, they utilize an existing diversification algorithm from prior work [51] that we use as a baseline. In summary, existing techniques return top-k unionable tables whereas, we output k unionable and diverse tuples for a given query table.

**Search result diversification.** The information retrieval literature considers diversifying search results (aka novelty search) [1, 4, 50]. They generally input a relevant item set to a query and output a subset of items that maximize a diversity score given by a diversification function [55]. The diversification function is defined by considering a trade-off between relevance and diversity that is controlled using a user-provided parameter, where increasing relevance decreases diversity and vice-versa. Furthermore, they are based on different objectives such as maximizing the sum of distances between the selected items (max-sum diversification) [3, 4, 22, 51], maximizing the minimum distance between the selected items (max-min diversification) [33], retrieving items in a diverse set having distances greater than a given threshold (threshold-based diversification) [9], and more. Note that selecting k-diverse items from a given set is an expensive problem [6, 51]. Therefore, various greedy algorithms are proposed to compute an approximate solution. For instance, Yu et al. [54] proposed the SWAP algorithm considering max-diversification of the items that a recommender system may suggest. SWAP starts by generating a candidate set and then greedily exchanges items in the candidate set to enhance the diversity score. Other work selects diverse sets by approximating diversity scores using different strategies [16, 22, 49]. Noticeably, Vieira et al. [51] developed GNE and GMC algorithms using a Maximum Marginal Relevance (MMR) scoring function that approximates the max-diversification score to give better diversification results than prior work. We use these algorithms

in the experiments. Allan et al. [3] provide theoretical guarantees and approximation bounds for the computation of MMR, which is central to Max-Sum diversification algorithms. Moreover, Drosou and Pitoura [9] proposed a threshold-based definition of diversity where they consider two items to be similar if they are within a given threshold. They develop an approximate algorithm to select a set of diverse items from a given set of relevant items such that each relevant item is represented in the diverse set by a similar item and each item in the result set is dissimilar to each other. Note that the diverse set could be empty if no items satisfy the given threshold. Hence, we develop a diversity algorithm to output k-diverse items instead of using a threshold-based approach. Note that all of this work considers diversification of image search, web search, and product recommendations. The diversification of tuples, specifically a set of unionable tuples, has not yet been studied. These prior IR diversification works assume relatively small sets of relevant items to diversify and consider effectiveness over efficiency as their use case is to display a small number of diverse search results or recommend a small number of diverse items for human consumption [5, 15, 50]. However, since we may need to yield hundreds of diversified tuples from tens of thousands of unionable tuples, we consider how to scale diversification algorithms.

Recently, Large Language Models (LLMs) have shown promising performance in different tabular tasks that need semantic understanding such as column type annotation [29], understanding table unionability [39] and entity matching [34]. For tuple diversification, we may prompt an LLM [13] with a list of tuples and ask it to return k-diverse tuples. However, as we have to consider thousands of tuples, the current LLMs' input token limits may prohibit their use for this task. Nevertheless, we adopt an LLM baseline [12] for an effectiveness experiment in a small dataset.

## 3 Problem Definition and Overview

We denote a query table using  $Q$  and a set of data lake tables using  $\mathcal{D}$  where  $T \in \mathcal{D}$  represents a data lake table. A tuple from any data lake table is called a *data lake tuple* ( $t$ ) and a tuple from a query table is called a *query tuple* ( $q$ ). We use  $c$  to represent a column and accordingly, for Table  $T$  containing a tuple  $t_i$ , we use  $T.c_j$  and  $t_i[c_j]$  to represent its  $j_{th}$  column and the corresponding value of tuple  $t_i$  in column  $c_j$  respectively. Furthermore, let  $E(t_i)$  be an embedding (representation) of tuple  $t_i$  in an embedding space.

### 3.1 Tuple Diversity Score

Let  $\delta(\cdot)$  be a tuple distance function that inputs embeddings of two tuples and outputs a distance between them such that the distance between embeddings of a tuple and itself is 0. Let,  $div(\cdot)$  be a diversity scoring function that maps embeddings of a set of tuples to their diversity score. We consider  $div(\cdot)$  to be the maximum for the optimal set of diverse tuples. In our evaluation, we will use common diversity functions from the literature that include maximizing the embedding distance between all tuples (Max-sum diversification) [51] or maximizing the minimum distance between tuples (Max-min diversification) [33].

### 3.2 Problem Definition

Given a set of  $n$  query tuples, the Diverse Unionable Tuple Search Problem aims to find a set of  $k$  unionable tuples in a data lake that are also diverse. We adopt the existing unionability definition that

two tuples are *unionable* if they are either from the same table or two unionable tables, i.e., tables sharing unionable columns [11, 37].

**DEFINITION 2 (DIVERSE UNIONABLE TUPLE SEARCH PROBLEM).** Given a Query Table  $Q$  having a set of tuples  $\{q_1, q_2, \dots, q_n\}$ , a set of unionable Data Lake Tables  $\{t_1, t_2, \dots, t_s\}$ , a positive integer  $k < s$ , and a diversity scoring function  $div(\cdot)$ , the diverse unionable Tuple Search Problem is: find a set of  $k$  unionable tuples such that  $div(\{q_1, q_2, \dots, q_n\} \cup \{t_1, t_2, \dots, t_k\})$  is maximized.

### 3.3 Solution Overview

Now, we present a high-level overview of DUST, our solution to the Diverse Unionable Tuple Search Problem (shown in Fig. 3 and Algorithm 1). Given a query table, DUST first obtains a set of unionable tables from the data lake, aligns their columns, and outer-unions the tables. A tuple embedding model is applied to encode each tuple. Then, DUST diversifies the tuples to select  $k$ -diverse tuples. Fig. 3 summarizes DUST pipeline which consists of three main steps and we now briefly introduce each step.

---

#### Algorithm 1: DUST

---

```

1 Input: Query Table  $Q$ , set of Data Lake Tables  $\mathcal{D}$ , Number of
  output tuples  $k$ 
2 //Discover data lake tables unionable with the query table.
3  $\mathcal{D}' \leftarrow \text{SearchTables}(Q, \mathcal{D})$ 
4 //Align matching columns of the discovered tables and union
  them.
5  $\mathcal{T} \leftarrow \text{AlignColumns}(Q, \mathcal{D}')$ 
6 //Embed each query and data lake tuple.
7  $E_Q, E_{\mathcal{T}} \leftarrow \text{EmbedTuples}(Q, \mathcal{T})$ 
8  $\mathcal{F} \leftarrow \text{DiversifyTuples}(E_Q, E_{\mathcal{T}}, k)$ 

```

---

**Creating Unionable Tuples.** We first explain the top half of Fig. 3, where we are provided with a Query Table  $Q$ . We search for unionable tables from the data lake using any table union search technique [11, 20, 24, 37]. To union these tables (to form unionable tuples), we perform column alignment. As part of the union search, many search algorithms will align each data lake table individually with the query table, e.g., Starmie uses maximum-weight bipartite matching between each column in a unionable table and the query table [11]. However, most methods do not output column alignments. Rather than recomputing pairwise matches, DUST uses a *holistic column matcher* [41] inspired by a recent data integration approach [26]. Holistic matching allows a collective alignment of all the columns in the set of unionable tables and the query table. Unlike in pure integration however [26] (where the goal is to integrate a set of tables), in our setting we want to do a targeted alignment to the query table. We are not interested in columns from data lake tables that align with each other if there is no query column to which they align. Our objective is to get a disjoint set of columns from the unionable tables such that the columns in each set are aligned together and to a single query column.

Similar to Khatiwada et al. [26], we first embed all query and data lake columns using the set of column values. Over such embeddings, we apply hierarchical clustering to generate a dendrogram that models all possible clusters of columns. Note that no two columns from the same table should be aligned together. Therefore, we enforce a constraint during clustering that prohibits clustering columns from the same table together. Then, an important decision is to select the number of clusters. For that,

we compute a cluster quality score for each number of clusters and select the one that maximizes the quality. We measure quality using Silhouette’s coefficient [26, 44].

Note that we only need values from those data lake columns that align with at least one column of the query table. So, after getting the clusters, we discard those that do not contain a column from the query table. This leaves us with a set of clusters, each with a query column and data lake columns that are aligned. If a query column has no match with one or more of the unionable tables, outer-union will pad the unionable table with a null placeholder (e.g., *nan*) to create tuples that can be unioned with the query table. Using this alignment, we can outer-union all unionable tables to form a set of **unionable tuples**. We provide further details and a block diagram of column alignment phase in the technical report [28].

**EXAMPLE 3.** Consider Tables in Fig. 1 such that Table (a) is a query table and Tables (b), (c), and (d) reside in the data lake. Suppose we use a union search technique to find the top-2 unionable tables from the data lake. Then Tables (b) and (d) will be returned as the two most unionable tables with Table (a). Next, we input columns of Tables (a), (b), and (d) into the column alignment phase where we get five clusters of columns. Specifically, the first cluster contains three Park Name columns from Tables (a), (b), and (d). The second cluster contains two Supervisor columns from Tables (a) and (b), and Supervised By column from Table (d). The third cluster contains City column from Table (a) and Park City column from Table (d). The fourth cluster contains two Country columns from Tables (a) and (b), and Park Country column from Table (d). We also get a singleton cluster with Park Phone column from Table (d). Since the last cluster contains no columns from the query table, it is discarded. Then we are left with the first four clusters of columns that are assigned with query columns’ headers Park Name, Supervisor, City, and Country respectively.

**Tuple Representation.** After forming unionable tuples, we embed them into a high-dimensional space (bottom right of Fig. 3). We use a novel embedding model (described in Sec. 4) that maps a tuple to its fixed-dimension embedding. We embed the query table’s tuples using the same model and column ordering.

**Tuple Diversification.** Finally, we use the tuple embeddings to find  $k$ -diverse tuples. Along with the embeddings, we are given a diversity scoring function  $div(\cdot)$ , and positive integer  $k$ . We input them to a tuple diversification algorithm that return a set of  $k$ -diverse data lake tuples. For the most diverse set, the diversity score measured using  $div(\cdot)$  is maximized. In our experiments, we will use existing diversification algorithms from the literature along with a new scalable alternative that we introduce in Sec. 5.

## 4 Unionable Tuple Representation

While diversification is a well-studied topic, diversifying unionable tuples is not. To apply the solution overviewed in the last section, we need to be able to compute a meaningful distance measure that models diversity (more distant means more diverse). Thus, we now create an embedding space to represent tuples, allowing the computation of distances. The diversity literature [51, 52] considers relevance and diversity as opposite dimensions, i.e., increasing relevance is considered to reduce diversity and vice-versa. Using this lens, two similar unionable tuples should be closer to each other in the embedding space and less diverse than two different unionable tuples. For example,

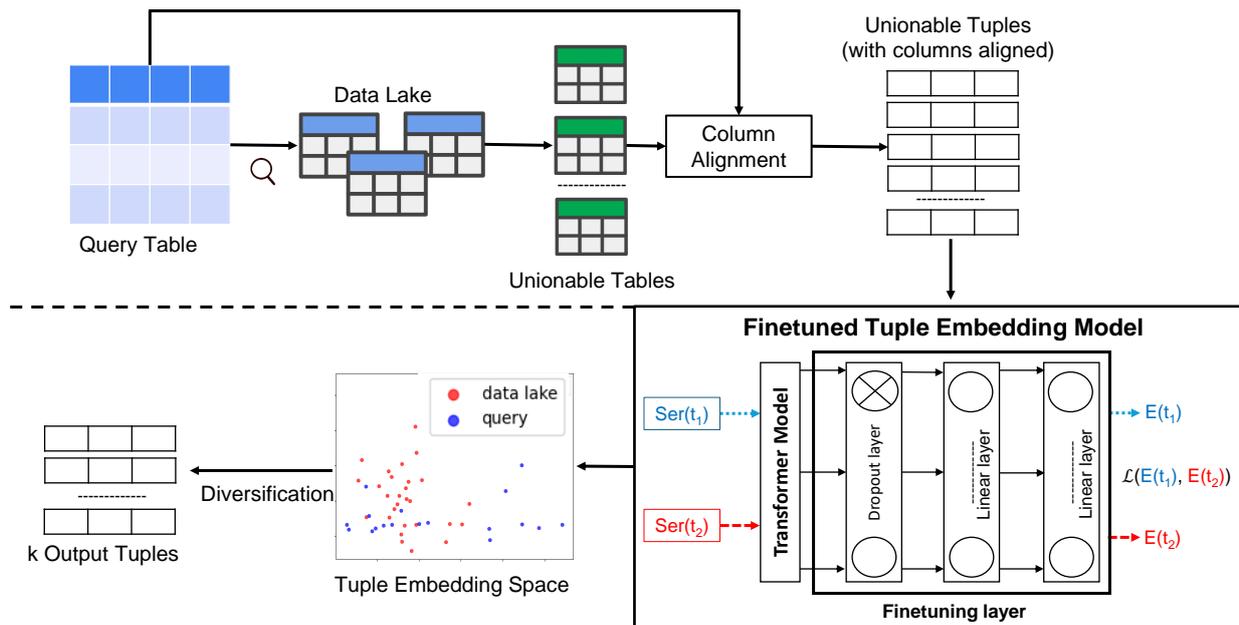


Figure 3: Block diagram of Diverse Unionable Tuple Search (DUST)

the tuple referring to River Park in Fig. 1 (a) should be much closer to the tuple referring to River Park in Fig. 1 (b) than the tuple referring to Chippewa Park in Fig. 1 (d). Now, since in this step we already assume the tuples are unionable, our goal would be to utilize this embedding space to position more distant, yet unionable tuples on the top of the diversity list. For example, given the aforementioned River Park tuple in Fig. 1 (a), we will use the embedding space to consider Chippewa Park in Fig. 1 (d) more diverse from the query table than the River Park tuple in Fig. 1 (b). In this case, both tuples are unionable, but as the latter is identical to a query tuple, the former should be preferred, being both unionable and more diverse.

To represent a tuple, it is important to consider what columns it contains and the context in which it appears. In related tasks such as entity matching [30], column annotation [48] and even table union search [11], pre-trained language models (e.g., BERT [7]) have been shown to correctly capture context. It would be reasonable to use them to capture an intuitive notion of similarity that comes from natural language (tuples using similar words will be closer in the embedding space). If we want to use an embedding model to embed unionable tuples for diversification, the model must understand if two tuples are unionable so that it can encode their distances accordingly. As our experiments indicate (Section 6), without proper fine-tuning, existing embedding models do not work well for capturing unionability. In what follows, we formulate a new fine-tuning technique aiming at classifying unionable tuples. Such a fine-tuned model could provide better performance by measuring how similar or diverse unionable tuples are. In addition to using an annotated table union search benchmark [37], we also use, in a self-supervision fashion, the fact the tuples originating from the same table are unionable. To that end, we create fine-tuned tuple embeddings, described next.

**Dataset Preparation.** We used a table union search benchmark [37] to create our training data. The benchmark contains labels indicating if two tables are unionable. Based on our classification task, we devise a fine-tuning dataset where each data point contains a pair of tuples and a binary label (0 if they originate

from non-unionable tables or 1 if they are from the same table or a pair of unionable tables). So for comprehensive fine-tuning, we create data points of similar tuples by selecting a pair of tuples from the same table or from two unionable tables. Furthermore, two tuples from a pair of non-unionable tables in the benchmark are about different topics and hence, diverse from each other. So, for the diverse tuple data points, we create tuple pairs by selecting a tuple each from two non-unionable tables. The data points are then divided into train, test, and validation sets without leakage.

**Serialization.** Here, we describe how we input the tuples to the pre-trained model for fine-tuning. Pre-trained language models are built over textual data and they take natural language sentences as input. In comparison to text, tabular data has different semantics and information encoding principles [8, 11, 48]. So, we serialize tuples into sentences to input them in the pre-trained model by retaining tabular properties. Specifically, two tuples may not have the same columns but may still be unioned on a subset of their columns. Hence, to help the model learn unionability on different numbers of columns, we serialize each tuple by separating each column and its value.<sup>1</sup> Precisely, let us consider a tuple  $t$  having  $n$  columns. Let  $c_i$  and  $v_i$  represent the column's header and value respectively of column  $i$  [7]. Also, [CLS] and [SEP] are the special BERT-based model tokens that represent the start of a sequence and a separation between the tokens in the sequence respectively. We feed the serialized tuple  $t$  to the input layer of the BERT-based model as:  $Ser(t) :- [CLS] c_1 v_1 [SEP] c_2 v_2 \dots [SEP] c_n v_n [SEP]$

**EXAMPLE 4.** Consider Tuples in Tables of Fig. 1. Recall from Example 3 that we assigned Park Name, Supervisor, City, and Country as column headers to the columns aligned with respective Query Columns (Table (a)'s columns). So, we serialize Tuple (River Park, Vera Oate, Fresno, USA) in Table (a) as: [CLS] Park Name River Park [SEP] Supervisor Vera Oate [SEP] City Fresno [SEP] Country USA [SEP]. Moreover, for Tuple (Chippewa Park, "Brandon, MN", USA, 773 731-0380, Tim Erickson) in Table (d), we

<sup>1</sup>We have experimented with other serializations offline, which were less effective.

use those columns (and order) for serialization that aligned with the Query Table, i.e., all but Park Phone. So, it is serialized as: [CLS] Park Name Chippewa Park [SEP] City Brandon, MN [SEP] Country USA [SEP].

**Fine-tuning Architecture.** We want to create a Tuple Embedding Space such that a pair of diverse tuples are farther from each other than a pair of similar tuples. Our empirical evaluation shows that the pre-trained models as-is are not good at this task (see Sec. 6.3). Therefore, as shown in Fig. 3 (bottom-right corner), we append a fine-tuning layer to the pre-trained transformer model and train it to embed the tuples such that the diverse tuples are far from each other and vice-versa. For fine-tuning, we test combinations of different layers and parameters empirically and use the best architecture. Specifically, in the DUST model, we append a dropout layer to the regular transformer model which possibly helps us avoid overfitting during training. Then, we pass the dropout output through two linear layers such that the outcome of the last linear layer is a fixed-dimension embedding of the tuple.

Note that we use the fine-tuned model to embed each tuple individually for diversification in the later phase (Sec. 5). So, during the training process, given a data point with a pair of tuples and a binary label, we pass each serialized tuple one after another through the model to represent them independently. As shown in Fig. 3 (bottom right) for tuples  $t_1$  and  $t_2$ , we pass  $Ser(t_1)$  and  $Ser(t_2)$  one after another to get their representations  $E(t_1)$  and  $E(t_2)$  respectively. Then, we compute loss by comparing the distance between two representations against the ground truth labels. The loss is fed back to update the model parameters. Our architecture is flexible to use any distance function. For our experiments, we use cosine distance, and accordingly, measure cosine embedding loss,

$$\mathcal{L}(E(t_1), E(t_2)) = \begin{cases} 1 - \cos(E(t_1), E(t_2)) & \text{if label} = 1 \\ \max(0, \cos(E(t_1), E(t_2))) & \text{if label} = 0 \end{cases},$$

where  $\cos(E(t_1), E(t_2)) = \frac{E(t_1) \cdot E(t_2)}{\|E(t_1)\| \|E(t_2)\|}$ .

## 5 Scalable Tuple Diversification

As we discussed, tuple diversification is computationally hard [51] and has traditionally been applied in IR to smallish sets (which have less than a few hundred of elements). Because we wish to consider in our experiments larger sets (with thousands of elements), here we propose a clustering-based approach that efficiently computes an approximate solution. We also propose a pruning method that controls the number of candidate diverse tuples input to the clustering algorithm. Our algorithm (Algorithm 2) takes as input two separate sets of embeddings for the unionable data lake tuples and query tuples, along with a positive integer  $k$ , and it returns  $k$ -diverse unionable data lake tuples. Recall that we want those data lake tuples that provide potentially new information to the query table. We consider two things when selecting the diverse tuples. First, we want to select a set of unionable data lake tuples such that the selected tuples are diverse among themselves. This helps us to ensure that the newly added tuples do not bring redundancy among themselves. Second, we ensure that the selected tuples are diverse from the set of tuples present in the query table. Algorithm 2 first prunes the set of unionable tuples (Line 2 and Sec. 5.1). Then, we compute

---

### Algorithm 2: DiversifyTuples

---

```

1 Input: set of query tuples embeddings  $E_Q$ , set of unionable data
   lake tuples embeddings  $E_{\mathcal{T}}$ , positive integer  $p$ , Number of
   output tuples  $k$ , Number of unionable tuples  $s$ 
2  $E_{\mathcal{T}'} \leftarrow \text{PruneTuples}(E_{\mathcal{T}}, s)$ 
3 //Cluster data lake tuples and select candidates.
4  $E_{\mathcal{T}'} \leftarrow \text{ClusterTuples}(E_{\mathcal{T}'}, k \cdot p)$ 
5 //Compute ranking score for each candidate data lake tuples
6  $\text{rank\_scores} \leftarrow ()$ 
7 for  $t \in \mathcal{T}'$  do
8    $\text{rank\_scores}[t].\text{insert}(0)$ 
9 for  $t \in \mathcal{T}'$  do
10  for  $q \in Q$  do
11     $\text{rank\_scores}[t].\text{update}(\min(\text{rank\_scores}[t], \delta(t, q)))$ 
12 //Return top-k data lake tuples
13  $\mathcal{F} \leftarrow \text{FindTopK}(\text{rank\_scores}, k)$ 

```

---

candidate unionable tuples that are diverse among themselves using clustering (Line 4 and Sec. 5.2). We compare these candidates against the query tuples and re-rank them such that the selected unionable tuples are diverse from the query tuples (Line 13 and Sec. 5.3).

### 5.1 Pruning Candidate Data Lake Tuples

Each unionable data lake table can have a large number of tuples and the step of clustering them within our diversification algorithm can be time-consuming. So, we begin by pruning and restricting the number of tuples before initiating clustering (Line 2). Note that we want to output data lake tuples that are diverse from each other. Specifically, when a tuple's embedding is significantly distant from others in the table, it signifies greater diversity than its counterparts. We initially compute the mean embedding of all eligible data lake tuples within each table for diversification. Then, for every tuple, we compute its distance from this mean embedding and rank them accordingly. Mathematically, for a tuple  $t$  from Table  $T$  having  $E(t_m)$  as the mean embedding of its tuples, the rank score of  $t$  is computed as  $\text{Score}(t) = \delta(E(t_m), E(t))$ . The top- $s$  tuples based on this ranking are then selected for clustering. As our pruning method prioritizes tuples with the highest distances from each other, it ensures the retention of the most diverse candidates for further processing.

### 5.2 Clustering Candidate Data Lake Tuples

To select a set of diverse candidate unionable tuples, we want to select candidates that are far from each other in the embedding space. For this, we apply clustering over the set of unionable data lake tuples. Precisely, we use hierarchical clustering as it can scale well for a reasonable number of clusters (in our case the number of output diverse tuples,  $k$ ). To ensure that we select enough candidates, we use a parameter  $p$  to control the number of clusters such that there are more than  $k$  candidate tuples. In Algorithm 2, the number of clusters (size of candidate tuple set)  $\mathcal{T}' = k \cdot p$  (see Line 4). As each cluster contains similar tuples and the tuples in different clusters are diverse, we select a representative diverse tuple from each cluster. Specifically, to increase the distance between candidate tuples, we select each cluster's medoid, which is the central-most element of the cluster. The medoids then form a candidate tuple set containing data lake tuples that are diverse among themselves. Selecting the medoid of each cluster as a candidate diverse tuple makes the approach more robust to outliers and this could also be augmented with an outlier detection phase before the clustering [46].

<sup>2</sup>In our experiments, we use PyTorch's implementation (<https://pytorch.org/docs/stable/generated/torch.nn.CosineEmbeddingLoss.html>).

### 5.3 Re-ranking Candidate Diverse Tuples

Among the candidate data lake tuples that are diverse among themselves, now we want to select tuples that are most diverse with respect to the query tuples. A candidate unionable data lake tuple can be diverse with one query tuple but not with other query tuple. Hence, we want to ensure that a selected unionable tuple is not redundant with any query tuples. For each candidate unionable tuple, we compute its distance from each query tuple and then assign a score which is its minimum distance from all the query tuples (Line 6-11 of Algorithm 2). We then sort the candidate tuples in descending order such that the best-ranked tuple has the maximum minimal distance from the query tuples. In case of a tie, we prioritize the tuple with the highest average distance from all query tuples. Finally, the set of top-ranked  $k$  unionable tuples in the ranking is returned as output (Line 13).

	$q_1$	$q_2$	$q_3$	Rank Score	Tie-breaking Score	Rank
$t_1$	0.3	0.1	0.9	0.1	0.43	4
$t_2$	0.5	0.4	0.6	0.4	0.5	1
$t_3$	0.75	0.5	0.1	0.1	0.45	3
$t_4$	0.4	0.55	0.5	0.4	0.48	2
$t_5$	0.9	0.75	0.01	0.01	0.55	5
$t_6$	0	0.99	0.2	0	0.39	6

Figure 4: Ranking candidate diverse unionable tuples.

**EXAMPLE 5.** Consider query tuples  $q_1, q_2$  and  $q_3$  and data lake tuples  $t_1, t_2 \dots t_6$ . Fig. 4 shows the tuple distance between each query and data lake tuple, and we want to rank the candidate tuples. First, we compute the Rank Score for each candidate tuple which is the minimum of its distance to all the query tuples. We also compute the average distance between each candidate tuple and query tuples to break ties (Tie-breaking Score). Tuples  $t_2$  and  $t_4$  have the highest Rank Score i.e. 0.4. Since both tuples have equal score, we look at Tie-breaking Score and rank  $t_2$  (score of 0.5) above  $t_4$  (score of 0.48). Similarly, Tuple  $t_3$  is ranked before Tuple  $t_4$ . Tuples  $t_5$  and  $t_6$  with the least Rank Scores (0.01 and 0) are ranked at the bottom.

### 5.4 Tuple Diversification Evaluation

In the literature, the objective of diversification is to simply maximize diversity among all results. Hence, algorithms are evaluated mainly on two criteria: Max-sum diversification where the sum of distances between items (tuples) in the diverse set is maximized [3], and Max-min diversification where the minimum distance between the diverse items in the set is maximized [33, 53]. However, for diverse unionable tuple search, we have a different objective: select unionable tuples that are *as diverse as possible* with query tuples and *as diverse as possible* to each other. So, based on those criteria, we present two adapted metrics to evaluate tuple diversification.

**(i) Average Diversity.** To evaluate if the distance between tuples is maximized (Max-sum diversification criteria), we compute the average distance between tuples in a diverse set. Precisely, we compute the average distance between query table tuples and the data lake table tuples, and between data lake table tuples. An optimal algorithm maximizes the distance between the tuples and hence gives the highest average diversity score. Given a set of Query Tuples  $\{q_1, q_2 \dots q_n\}$  and a set of  $k$ -diverse unionable data lake tuples  $\{t_1, t_2 \dots t_k\}$  returned by a method, then the Average Diversity is computed as:

$$\text{Average Diversity} = \frac{\sum_{i=1}^n \sum_{j=1, i \neq j}^k \delta(q_i, t_j) + \sum_{i=1}^{k-1} \sum_{j=i+1}^k \delta(t_i, t_j)}{n+k} \quad (1)$$

As query tuples are given and the distance between them is constant for all algorithms, we exclude their distance from the evaluation.

**(ii) Min Diversity.** Based on the Max-min Diversification criteria, i.e., maximizing the minimum distance between the items in the diverse set, we also compare the minimum distance among the tuples selected in a  $k$ -diverse set. An optimal algorithm gives the highest Min Diversity score. Mathematically, for query tuples  $\{q_1, q_2 \dots q_n\}$  and a set of  $k$ -diverse unionable data lake tuples  $\{t_1, t_2 \dots t_k\}$  returned by a method, then  $S_Q = \{\delta(q_i, t_j) \mid i \in [1, n], j \in [1, k]\}$ ,  $S_T = \{\delta(t_i, t_j) \mid i \in [1, k-1], j \in [i+1, k]\}$ ,

$$\text{Min Diversity} = \min\{S_Q \cup S_T\} \quad (2)$$

Prior work [51] also considers the trade-off between relevance and diversity controlled by a user-defined parameter. Accordingly, they formulate a diversity scoring function that inputs a set of  $k$ -diverse items and returns their diversity score. The optimal set of  $k$ -diverse items gives the maximum diversity score. As determining such an optimal set is computationally hard, they created ground truth considering  $k = 5$ , by using a brute force approach over 5 different queries each having 200 candidate relevant items. They report precision and the gap between the diversification scores of the diverse set returned by an algorithm and the ground truth set. In our case, we are looking to rank thousands of tuples and thus such an evaluation framework is not feasible.

## 6 Experiments

We now empirically evaluate DUST against different baselines.<sup>3</sup> We run all our experiments using Python 3.8 on a server with Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz processor and  $4 \times 8$  GB Tesla GPUs. We start by describing the benchmarks that we use for our evaluation (Sec. 6.1). Next, we investigate column alignment (Sec. 6.2) using several different existing column embeddings and comparing our holistic approach to a state-of-the-art pairwise matcher from Starmie [11]. Then, we aim to answer the following: **RQ1.** As a sanity check, we first consider how effective is DUST in distinguishing unionable tuples against existing tuple embedding techniques (Sec. 6.3)? **RQ2.** How well do existing diversification algorithms perform when diversifying unionable tuples? Do the new more scalable DUST diversification algorithm achieve the goal of scaling diversification to larger numbers of tuples and if so, what is the impact on effectiveness (Sec. 6.4)? **RQ3.** How effective and efficient is DUST in the end-to-end task of finding diverse unionable tuples in comparison with existing unionable table search techniques (Sec. 6.5)? **RQ4.** Does DUST provide practical and intuitive benefit over existing unionable table search techniques (Sec. 6.6)?

As each experimental setup and baselines differ across research questions, they are discussed within the corresponding subsections.

### 6.1 Benchmarks

We experiment using table union search benchmarks from the literature [24, 37, 39], each comes with a set of query tables and a labeled set of unionable data lake tables. Fig. 5 details the number of tables, columns, and tuples in each benchmark.

**6.1.1 TUS Benchmark [37].** The TUS benchmark contains over 5,000 tables generated by selecting and projecting rows and columns from 32 non-unionable base tables. The generated tables

<sup>3</sup><https://github.com/northeastern-datalab/dust>

Benchmark	Query			Data Lake			# Average Unionable Tables Per Query
	# Tables	# Columns	# Tuples	# Tables	# Columns	# Tuples	
TUS	125	1.6K	557K	5044	55.5K	9.6M	188
TUS-Sampled	30	355	134K	233	3.1K	1M	10
SANTOS	50	615	1.07M	550	6.3K	3.8M	14
UGEN-V1	50	400	550	1000	8K	10K	10

Figure 5: Benchmarks used in the experiments.

originating from the same base table are unionable, those from different base tables are non-unionable.<sup>4</sup> We create two variations of the TUS Benchmark based on our experimental requirements.

**TUS-Sampled Benchmark.** On average, the TUS Benchmark has over 188 unionable data lake tables per query table, significantly higher than other benchmarks. So to test effectiveness of some non-scalable methods, we created *TUS-Sampled* by selecting 30 query tables and sampling 10 unionable tables per query table.

**TUS Fine-tuning Benchmark.** To build the DUST tuple representation model (see Sec. 4), we create a fine-tuning benchmark using tables and unionability ground truth from TUS benchmark [37]. The created benchmark consists of 60K data points where each data point consists of a tuple pair and a unionability label (0 or 1) representing whether the tuples are unionable. The benchmark is balanced, i.e., it contains 30K unionable and non-unionable tuple pairs each. The tuple pairs selected from the same table or two unionable tables are considered unionable and the tuples from two non-unionable tables are considered non-unionable. We divide data points into train/test/validation sets in a popularly used ratio of 70:15:15 (42K, 9K, and 9K data points respectively). We ensure all three sets are balanced regarding the number of unionable and non-unionable tuple pairs. We also ensure that there is no data leakage between train, test, and validation sets.

**6.1.2 SANTOS Benchmark [24].** This benchmark contains 550 data lake tables and 50 query tables created by projecting and selecting rows and columns of 297 base tables from Canada, US, UK, and Australian Open Data.<sup>5</sup> The tables are created following the TUS benchmark creation approach. But unlike TUS, when creating the tables, SANTOS also considers the binary relationships between the column pairs such that the unionable tables not only contain unionable columns, but also share at least one binary relationship.

**6.1.3 UGEN-V1 Benchmark [39].** The UGEN-V1 benchmark is a table union search benchmark that is generated using a Large Language Model [12]. It contains 50 query tables generated from different topics and each table has 10 unionable data lake tables and 10 non-unionable tables on the same topic. As LLMs are not always accurate, the authors also provide a manually verified ground truth which we use in our experiments.<sup>6</sup>

Finally, we remove the columns having all null values and query tables that have less than 3 rows from our experiments.

## 6.2 Column Alignment Evaluation

For column alignment, we use existing embedding models to embed the columns. So, we empirically evaluate different embeddings for aligning unionable data lake columns with the query table columns.

<sup>4</sup><https://github.com/RJMillerLab/table-union-search-benchmark>

<sup>5</sup><https://zenodo.org/records/7758091>

<sup>6</sup><https://github.com/northeastern-datalab/gen>

**6.2.1 Experimental Setup.** Recall that to align columns, we first represent each column in an embedding space using pre-trained embedding models. Then, based on their embeddings, we cluster them into disjoint set of columns. So, we evaluate the performance of different embedding methods when they are used to represent the columns and align them. For clustering, we use the Agglomerative Clustering module available in Scikit-learn’s library.<sup>7</sup> We report results using average linkage and Euclidean distance based on empirical effectiveness. Following the literature [26], we use Silhouette’s Coefficient to measure the cluster quality [44].

**6.2.2 Evaluation Metrics.** Similar to the prior work [26], we report Precision (P), Recall (R), and F1-Score (F1) using different embeddings for column alignment. Recall that we discard clusters without a query column (see Sec. 3). Accordingly, we want to evaluate if the data lake columns are assigned to clusters with their correct aligning query column. The ground truth contains all true column alignments in the form of column pairs. This includes column pairs formed by each query column with its aligning data lake columns, and column pairs formed by two data lake columns that have the same matching query column. Furthermore, it is important to distinguish the query columns having no matching data lake columns. So, we also include each query column with no match in the ground truth. In the same way, we form a set of column alignments using the clusters produced by a method. Let,  $A_G$  be the set of true column alignments in the ground truth. Let,  $A_M$  be the set of column alignments given by a method. Then, we compute **P**, **R** and **F<sub>1</sub>** as:  $P = \frac{A_G \cap A_M}{A_M}$ ,  $R = \frac{A_G \cap A_M}{A_G}$ ,  $F_1 = \frac{2 \cdot P \cdot R}{P + R}$ .

**6.2.3 Baselines.** Now, we describe different embedding models that we test for the column alignment task. We use two word embedding models: **FastText** [23] and **Glove** [40] for our experiments. Furthermore, we also experiment using three language models: **BERT** [7], **RoBERTa** [31] and Sentence BERT (**sBERT**) [43]. For all models, we create a **Cell-level** variation where we embed a column by first representing each cell value independently and then averaging the representation of all the values within a column. Furthermore, for language model baselines, we also create a **Column-level** variation. Specifically, rather than treating each cell independently, the **Column-level** variation concatenates all cell values into a sentence and inputs it into the model. The model then returns a representation for the column. Note that all three language models have a token limit of 512. Therefore, using all the column values in the input may not be possible. Hence, we follow the literature [8, 11, 48] and select at most 512 most representative tokens for each column based on their TF-IDF scores [42]. Moreover, table union search techniques such as Starmie [11], assess table unionability based on the maximum-weight bipartite matching between query and candidate data lake tables’ columns. Starmie embeds each column to capture the entire **Table context**. So, we use Starmie as a baseline, implementing two versions: **Starmie (B)**, by applying *bipartite* matching between the columns embedded using Starmie. Next, we also implement **Starmie (H)** that uses our *holistic* column alignment approach on Starmie’s column embeddings.

**6.2.4 Effectiveness.** We now evaluate the effectiveness of different embedding methods on the column alignment task. Specifically, we report precision, recall, and F1-Score using each embedding method in Table 1. For our discussion, we focus on the

<sup>7</sup><https://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster>

**Table 1: Column Alignment effectiveness. Best score along each column is in bold; the second best score is underlined.**

Serialization	Model	TUS-Sampled			SANTOS			UGEN-V1		
		P	R	F1	P	R	F1	P	R	F1
Cell-level	FastText	0.86	0.60	0.66	0.64	0.86	0.70	0.36	0.78	0.43
	Glove	0.59	0.83	0.63	0.65	0.84	0.71	0.40	0.76	0.43
	BERT	0.60	0.60	0.59	0.57	0.68	0.60	0.41	0.61	0.44
	RoBERTa	0.61	0.80	0.69	0.57	0.84	0.66	0.52	0.68	0.53
	sBERT	0.67	0.77	0.70	0.60	0.86	0.69	0.50	0.71	0.52
Column-level	BERT	0.80	0.57	0.64	0.68	0.68	0.66	0.49	0.57	0.47
	RoBERTa	0.81	0.72	<b>0.74</b>	0.71	0.87	<b>0.76</b>	0.58	0.66	<b>0.58</b>
	sBERT	0.74	0.72	0.68	0.71	0.86	0.76	0.54	0.71	0.58
Table context	Starmie (B)	0.30	0.67	0.41	0.23	0.53	0.32	0.15	0.76	0.24
	Starmie (H)	0.83	0.43	0.55	0.43	0.33	0.18	0.64	0.62	0.57

combined metric i.e., F1-Score (highlighted in gray). We observe that the Column-level RoBERTa performs the best in aligning the columns as it has the highest F1-score in all the benchmarks. Specifically, RoBERTa outperforms the second-best method, the Column-level variation of sBERT, by around 8% in TUS-Sampled. In SANTOS and UGEN-V1 benchmarks, sBERT also performs as well as RoBERTa. BERT, possibly because it is the smallest among the three language models, has the lowest F1-score among them. Further, we observe that for all three language models, the Column-level variation is more effective than the Cell-level variation in terms of F1-score in almost all cases; the only exception is sBERT in TUS-Sampled by a small margin. A possible reason for Column-level variation’s better performance is that the Column-level variation gets more tokens as input at once, and accordingly, it becomes easier for them to understand the column than for Cell-level variation where they receive tokens only from one cell at a time to understand the context. Moreover, the word embedding baselines (FastText and Glove) give comparative performance to cell-level language models in all the benchmarks in terms of F1-score, but they are comprehensively outperformed by column-level RoBERTa and column-level sBERT. This also shows that larger language models can contextualize the columns better to align them. Additionally, both variations of Starmie are mostly outperformed by other models. This may be attributed to Starmie embedding each column with the context of the entire table, resulting in columns from the same table having closer representations. However, column alignment requires contextual understanding specific to the column itself, which differs from other columns within a table. Worth noting, Starmie (bipartite), which matches columns of a table pair, is generally outperformed by Starmie (holistic). Our holistic matching approach considers a broader context by aligning a set of tables together, resulting in improved performance. The SANTOS benchmark contains a higher proportion of numerical columns compared to other benchmarks, which are not effectively embedded by Starmie. As a result, the holistic approach tends to create distinct clusters for numerical columns, leading to a decrease in recall and subsequently, the F1 score. In conclusion, our holistic approach consistently achieves superior column alignments than bipartite matching with well-embedded columns. Accordingly, DUST uses the best-performing Column-level RoBERTa model.

**6.2.5 Efficiency.** Next, we report runtime to align the columns in each benchmark. Note that this time depends on the time taken by the model to embed the columns and run the clustering algorithm. Since the embedding time for each model is

significantly fast (on average, less than a second per column in our benchmarks) and they are insignificantly different, Column Alignment time is dominated by the clustering time. On average, the column alignment time per query is reasonable (35, 46, and 24 seconds in TUS-Sampled, SANTOS, and UGEN-V1 Benchmarks respectively).

### 6.3 Tuple Representation Evaluation

Now, we compare the performance of DUST tuple representation against state-of-the-art baselines.

BERT	RoBERTa	sBERT	Ditto	DUST (BERT)	DUST (RoBERTa)
0.50	0.50	0.56	0.66	0.84	<b>0.85</b>

**Figure 6: Unionable tuple representation Accuracy. Best score is bolded and the second best is underlined.**

**6.3.1 Experimental Setup and Evaluation Metric.** To evaluate DUST’s embedding model against other embedding models, we report **Accuracy** computed over the test set of TUS Fine-tuning Benchmark (Sec. 6.1). Each data point is a pair of tuples and an embedding model returns embeddings for each of them. We then compute their cosine distance. If the cosine distance is less than a threshold, we consider the tuple pairs predicted to be unionable. Otherwise, the tuple pairs are predicted to be non-unionable. Based on empirical evaluation, we use a threshold of 0.7 which gives the best accuracy in the validation set. We compute Accuracy using Scikit-learn’s implementation.<sup>8</sup> Mathematically, let  $TP, FP, TN, FN$  be the number of true positives (unionable tuple pairs predicted as unionable), false positives (non-unionable tuple pairs predicated as unionable), true negative (non-unionable tuple pairs predicted as non-unionable) and false negative (non-unionable tuple pairs predicated as non-unionable). Then,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

**6.3.2 Baselines.** We now explain the baselines that we use for the experiments. Since there are no models for embedding unionable tuples, we adapt transformer-based embedding models that are successful in other semantic tasks. Specifically, we use pre-trained **BERT** [7] and **RoBERTa** [31] models available in huggingface package.<sup>9</sup> Furthermore, Sentence BERT (**sBERT**) [43] is a recent state-of-the-art transformer-based model that is fine-tuned to closely embed sentences having similar meanings. We use sBERT

<sup>8</sup>[https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html)

<sup>9</sup><https://huggingface.co/models>

as another baseline to test if it can be fine-tuned to understand unionable tuples.<sup>9</sup> For consistency, we use the same serialization for all the models as we explained in Sec. 4. Furthermore, entity matching is also a similar classification task where the objective is to test whether the two input tuples are about the same real-world entity. So, to evaluate if entity matching techniques can understand tuple unionability, we also compare the DUST model against **Ditto**, a transformer-based model fine-tuned for the entity matching task [30]. We implement Ditto using their public code.<sup>10</sup>

**6.3.3 DUST models.** We build two variations of DUST. First we fine-tune BERT [7] (**DUST (BERT)**). Next, we fine-tune RoBERTa [31] (**DUST (RoBERTa)**). Note that both variations use the same serialization and the same fine-tuning architecture as described in Sec. 4. We keep embedding dimension equal to 768 which is consistent with the pre-trained models. We train each model for at most 100 epochs. However, to avoid model overfitting, we apply an early-stopping strategy [56] with a patience of 10. Specifically, we compute the validation loss after each epoch and if the loss does not improve for 10 epochs, we terminate the training process and consider the model to be converged. It took around 30 hours to fine-tune both **DUST (BERT)** and **DUST (RoBERTa)** using our experimental setup (four 8 GB GPUs). Note that for dynamic data lakes, this fine-tuning can be optimized using more modern hardware and applied periodically when a lake has changed significantly. We share our fine-tuned models and training parameters.<sup>3</sup>

**6.3.4 Effectiveness.** We report the accuracy of each baseline and the two variations of DUST in Fig. 6. DUST (RoBERTa) performs slightly better than its BERT-based variation. A possible reason for this is that it is pre-trained over a larger number of parameters than BERT and has larger prior knowledge. Consequently, we use DUST (RoBERTa) for all our experiments and to compare against other baselines. Next, we compare DUST’s performance against the baselines. It is seen that DUST outperforms the best baseline, Ditto, by over 15% showing that embedding tuples to understand unionability is different than embedding them for entity matching. Moreover, DUST outperforms sBERT by over 28% indicating that models that are good at detecting similar sentences can not capture intrinsic tabular properties and hence, they are not effective in determining tuple unionability. Further, we observe that both the pre-trained models, BERT and RoBERTa, perform as well as only tossing a coin, and are not able to understand tuple unionability. So, the low effectiveness of pre-trained models and fine-tuned models for embedding tuples for other tasks empirically validate the need to build DUST model to embed unionable tuples. Also, DUST embeddings are robust towards the change in column position within tuples, which we validate empirically in the technical report [28].

## 6.4 Tuple Diversification Experiments

Next, we evaluate the performance of our novel tuple diversification algorithm against state-of-the-art baselines (RQ2).

**6.4.1 Experimental Setup and Evaluation Metrics.** As we highlighted in Sec. 5.4, we use two tuple diversification measures, namely, Average Diversity and Min Diversity. To keep the distance function consistent with the distance function that we use in the Tuple Representation Model (see Sec. 4 and Sec. 6.3), we

**Table 2: Reporting (i) the number of query tables for which each diversification algorithm gives the best Average and Min diversity scores and (ii) Average Time taken per Query by each algorithm in each benchmark.**

Method	SANTOS			UGEN-V1		
	# Average	# Min	Time (s)	# Average	# Min	Time (s)
GMC	23	1	556	3	2	<1
GNE	-	-	-	0	0	81
CLT	0	0	82	18	12	<1
DUST	27	49	85	27	34	<1

use cosine distance<sup>11</sup> to measure the distance between the tuples throughout the experiments. Note that experiments with other distances such as Manhattan distance and Euclidean Distance are available in the DUST github. With both distances, the relative performance of all the baselines is similar to using the cosine distance. In addition, we run an analysis to select  $p$  in Algorithm 2. We measure the improvement in the diversity metrics (Sec. 5.4) when we increase the value of  $p$ , i.e., when we increase the number of candidate data lake tuples. Since, the improvement in the diversity metrics is either negative or insignificant for  $p$  more than 2, we select  $p = 2$  in our experiments. We provide further details in the technical report [28].

**6.4.2 Baselines.** We compare the DUST diversification algorithm with several state-of-the-art baselines from IR. Specifically, Vieira et al. [51] proposed different diversification algorithms and they have been adopted by a recent work [32] to search for tables having diverse sets of columns. So, we compare with the best-performing algorithms presented by Vieira et al. [51]. For each baseline, we use the default parameters suggested in the respective papers. We reproduced all algorithms following the original papers.

**GMC [51].** GMC (Greedy Marginal Contribution) is used to diversify search results. It considers a trade-off between relevance and diversity based on a user’s preference and formulates a diversity scoring function. Then, it greedily selects items one after another to add to the result set. An item is selected if it increases the diversity score computed using the items currently in the result set.

**GNE [51].** GNE (Greedy Randomized with Neighborhood Expansion) is a randomized version of GMC algorithm. It starts by creating a candidate diverse set following the same diversity scoring function as GMC. Then it iterates over the candidate set to replace a random item with other items not included in the candidate set.

**CLT [49].** Since we use a clustering approach in our algorithm to retrieve candidate diverse data lake tuples, we also compare against a simple clustering-based baseline that was used to select diverse images. CLT generates  $k$  clusters and selects an item from each cluster using different strategies. To keep experiments consistent, we select each cluster’s medoid in the diverse set. Further, we use the same clustering technique as ours with the same parameters.

**6.4.3 Effectiveness.** We run experiments with  $k = 100$  in the SANTOS benchmark, and  $k = 30$  in the UGEN-V1 benchmark. For each benchmark, we experiment with  $s$  (number of candidate unionable tuples) equal to at most 2500. This selection is based

<sup>10</sup><https://github.com/megagonlabs/ditto>

<sup>11</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine\\_distances.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_distances.html)

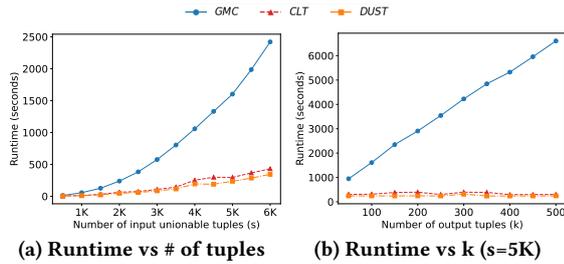


Figure 7: DUST Diversification Runtime against baselines

on the number of unionable tuples per query available in the data lake of each benchmark and also to accommodate baselines that do not scale for large  $k$  and  $s$ . Nevertheless, for efficiency experiments, we will report results for larger  $k$  using a synthetic dataset (Sec. 6.4.4). Note that the TUS benchmark is not used in this experiment as it was used to build the DUST model.

Our benchmarks have multiple query tables and experiments on each query are independent of one another. So, following the literature [51], we report Average Diversity and Min Diversity Scores by each baseline on each query table separately. Specifically, we report the number of queries for which each technique gives the highest Average Diversity and the highest Min Diversity Scores in comparison to other baselines (Table 2). The actual scores are provided in the github.<sup>3</sup>

We first run an experiment using a simple **random** baseline, sampling  $k$  tuples for each query and calculating their diversity scores. For a comprehensive analysis, we generated five random sets for each query using different random seeds and selected the highest-performing random set for each metric for comparison against DUST. In the SANTOS benchmark, DUST outperformed the random baseline in 46 out of 50 queries for Average Diversity and in all but one query for Min Diversity. In the UGEN benchmark (smaller tables), random is better than DUST only for less than 25% of queries for Average Diversity and in no queries for Min Diversity. These results show that random sampling is ineffective for tuple diversification. We now compare against the baselines from the literature.

DUST achieves the best Average Diversity scores for over 54 % of the queries in the SANTOS benchmark, outperforming the second best method, GMC, by 8 % (Table 2). DUST is the best method for more than half queries in UGEN-V1 benchmark as well, where the second-best method, CLT achieves the best performance for around 36% of queries. Interestingly, for all queries in the SANTOS benchmark where GMC performs the best, and for over 88 % of queries in the UGEN-V1 benchmark where CLT performs the best, DUST is the second-best method which shows that DUST is more effective than the baselines in diversifying the tuples. As GNE does not scale to large datasets, we evaluate it only in the UGEN-V1 benchmark where it is outperformed by all the baselines. In terms of Min Diversity, DUST gives the best performance for almost all queries in the SANTOS benchmark and around 70% of queries in the UGEN-V1 benchmark. This shows that first selecting candidate data lake tuples and then ranking them based on their distance from the query tuples improves diversification.

**6.4.4 Efficiency.** Now we compare DUST’s scalability against the baselines. In the SANTOS benchmark, DUST is over 7 times faster than GMC and almost as fast as CLT on average (Table 2). In the UGEN-V1 benchmark, which has fewer unionable tuples

per query than SANTOS, all but the GNE method diversify tuples within a second per query on average. The GNE algorithm, which first generates a candidate diverse set and then improves it iteratively, is much slower than all the baselines. Hence, in all the benchmarks, DUST gives the best effectiveness being much faster than the second-best baseline (GMC) and as fast as other baseline (CLT).

To understand how the number of input unionable tuples ( $s$ ) and the number of output tuples ( $k$ ) impacts the runtime, we experiment with a query table and a variable number of tuples that are unionable with the query table. Specifically, we first vary  $s$  for constant  $k$  ( $k = 100$ ) and report the runtime (Fig. 7 (a)). We saw that the increase in the number of input tuples increases the runtime for GMC quadratically whereas DUST is the fastest for which the runtime is linear to  $s$  with a very small slope. Also, when we vary  $k$  for  $s = 5000$ , and observe its impact on the runtime (Fig. 7 (b)), it is seen that DUST is not impacted by the increase in  $k$ . Furthermore, although DUST has an additional step after clustering where it selects the unionable data lake tuples based on their distance from each query tuple, it has a similar runtime as a clustering baseline (CLT). Hence, DUST’s post-clustering phase is scalable practically. An analysis of pruning, showing its importance in reducing runtime, is provided in the report [28].

## 6.5 DUST Against Table Search Techniques

We now compare DUST against two state-of-the-art table union search techniques, along with an LLM approach to this problem. We report diversity scores (see Sec. 6.4.1) of  $k$ -tuples that DUST outputs against that of  $k$ -tuples returned by table search technique.

**6.5.1 Baselines.** We now describe our baselines.

**D3L [2].** D3L aggregates different column unionability signals, such as value match, word embedding match, regular expressions, and so on, to search for the top- $k$  unionable tables from a data lake. We implement D3L using its public code.<sup>12</sup>

**Starmie [11].** Starmie is the state-of-the-art table union search technique that returns a set of top- $k$  unionable tables, given a query table. To adopt Starmie for searching  $k$  unionable tuples, we index each tuple in the data lake as a separate table and search for the top- $k$  tables. As each data lake contains a single tuple, the tuples from the top- $k$  searched tables are the  $k$  output tuples.

**LLM [12].** We use a Large Language Model (GPT-3) as a baseline to generate diverse unionable tuples. We input query table tuples to GPT-3 and ask it to generate a set of  $k$  diverse unionable tuples with the given query table. The robustness and effectiveness of LLMs is impacted by the selected prompt and thus we have evaluated several prompts (the prompt is provided in the github).<sup>3</sup>

For a fair comparison with DUST, we embed the output tuples by each baseline using DUST embeddings and compute diversity scores over them. We implement Starmie using its public code with the default parameters.<sup>13</sup> We implement GPT-3 using its API [38]. Additionally, we implemented Ver [17], a Query-By-Example system that takes a small number of tuples as input and adds new data lake tuples to it.<sup>14</sup> However, due to the creation of large indexes, the experiment for VER could not be scaled across any benchmarks. For instance, in UGEN-V1 Benchmark

<sup>12</sup><https://github.com/alex-bogatu/d3l>

<sup>13</sup><https://github.com/megagonlabs/starmie>

<sup>14</sup><https://github.com/TheDataStation/ver>

**Table 3: Number of query tables for which each diversification algorithm performs the best.**

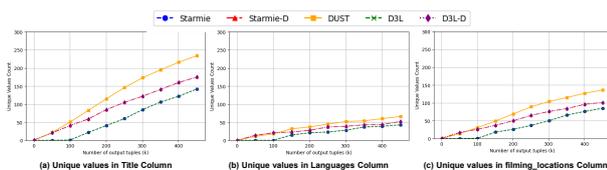
Method	SANTOS		UGEN-V1	
	# Average	# Min	# Average	# Min
Starmie	5	1	11	2
LLM	-	-	14	21
DUST	45	49	23	25

with 1000 data lake tables with a total of 8K columns and 10k rows (Fig. 5), Ver was automatically terminated after two days.

**6.5.2 Effectiveness.** We now report DUST’s diversification effectiveness against baselines in SANTOS and UGEN-V1 benchmarks. We exclude LLM from the analysis in SANTOS benchmark as it was not scalable for the query tables with a large number of tuples. Table 3 reports the number of queries in each benchmark where each method performs the best. The actual diversity scores are provided in the github.<sup>3</sup> In the SANTOS benchmark, DUST achieves the highest Average Diversity score for around 90% of queries and achieves the highest Min Diversity across almost all queries. This can be attributed to the baseline’s (Starmie’s) inclination to favor similar tuples, resulting in the retrieval of tuples already present in the query table. Further, in the UGEN-V1 benchmark, DUST consistently yields the most diverse tuples for the highest number of queries, outperforming the LLM, the second best baseline, by around 18% in terms of Average Diversity. Interestingly, for a given query, the LLM generates a few diverse tuples but subsequently, it produces redundant ones. Nevertheless, DUST could be scalable to search for 100s of tuples whereas LLM could not do so currently due to its token limits. Also, Starmie, whose Mean Average Precision (MAP) [24] of searching for unionable tuples in UGEN-V1 is 64%, shows better diversity performance than it does on the SANTOS Benchmark where its MAP is 78%. Starmie finds more non-unionable tuples in UGEN-V1 (lower MAP), and such tuples are generally more diverse with each other. These results support DUST’s necessity for diversifying unionable tuples to improve their usability.

## 6.6 Case Study

Finally, we show a case study, providing intuitive evidence of the benefits of diversification. We have created a small benchmark consisting of a query table and 20 unionable tables derived from an IMDB movie dataset [21]. The original IMDB table contains information on nearly 500 recent movies, including title, director, genre, budget, filming location, language, and more. We sample its rows to create a query table and unionable tables. On average, the tables in this benchmark have 97 tuples and 13 columns (see repository)<sup>3</sup>. Please note that this case study only aims to examine the diversity and thus only contains unionable tables/tuples.

**Figure 8: Number of novel values added to the different columns of query table by each method.**

Using DUST, we search for a set of  $k$  diverse tuples from this small data lake. We compare this to D3L and Starmie’s result.

Since both of the baselines output the top- $N$  unionable tables, we (bag) union the output tables based on their ranking with the query table until we have a bag with at least  $k$  tuples. From the resulting unioned table, we select  $k$  tuples (using SQL LIMIT  $k$ ) and compare to the  $k$  tuples produced by DUST. Starmie, D3L and other unionability methods [20, 24, 37] may find tables that overlap with the query table (and each other). So we also examine a duplicates-free version of D3L and Starmie (dubbed D3L-D and Starmie-D, respectively, in Fig. 8), in which we exclude duplicated tuples. Here, we take the set union of the top tables until this set contains at least  $k$  tuples and again if there are more than  $k$  we use the SQL LIMIT  $k$  on this set. Then, we measure and report how many new values each method adds across different columns of the query table in IMDB Benchmark. We refrain from comparing to an LLM baseline here as we aim to examine only movies from the given data lake.

In Fig. 8, we plot the number of output tuples ( $k$ ) on the X-axis and the number of unique values added by D3L, D3L-D, Starmie, Starmie-D, and DUST to the query table’s (a) Title, (b) Languages, and (c) Filming\_locations columns on the Y-axis. Statistics for other columns are similar and available in the repository.<sup>3</sup> Our results show that DUST retrieves tuples with nearly 25% more unique movie titles from the data lake than Starmie even when removing the duplicate tuples (Starmie-D). Interestingly, both D3L and Starmie add a similar number of unique values. This is because our evaluation focuses on a data lake composed exclusively of unionable tables. Since both baselines rely on table similarity to identify unionable candidates, their results tend to overlap as we discussed in Ex. 1. To illustrate DUST’s practical benefit even further, we also provide an anecdotal example showing the tuples it suggests against Starmie in the report [28].

## 7 Conclusion

We introduced the problem of finding diverse unionable tuples from a data lake and presented DUST as a first solution. We compared each component of DUST with relevant baselines and demonstrated its superior performance. We illustrated that DUST outperforms two state-of-the-art table union search techniques and a Large Language Model in terms of effectiveness and efficiency in discovering diverse unionable tuples. Our work suggests the need for human-in-the-loop discovery techniques that allow a user to interactively select diverse tuples of interest to them. We continue to explore augmenting our tuple embeddings together with additional "unembedded" tabular features such as column values and relationship between column pairs.

## Acknowledgments

This work was supported in part by NSF under award numbers IIS-1956096, IIS-2107248, and IIS-2325632. We acknowledge the support of the Canada Excellence Research Chairs (CERC) program. Nous remercions le Chaires d’excellence en recherche du Canada (CERC) de son soutien.

## Artifacts

The source code, data, and/or other artifacts have been made available at: <https://github.com/northeastern-datalab/dust>.

## References

- [1] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Trans. Knowl. Data Eng.* 17, 6 (2005), 734–749. doi:10.1109/TKDE.2005.99

- [2] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 709–720. doi:10.1109/ICDE48307.2020.00067
- [3] Allan Borodin, Aadhar Jain, Hyun Chul Lee, and Yuli Ye. 2017. Max-Sum Diversification, Monotone Submodular Functions, and Dynamic Updates. *ACM Trans. Algorithms* 13, 3 (2017), 41:1–41:25. doi:10.1145/3086464
- [4] Jaime G. Carbonell and Jade Goldstein. 1998. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, W. Bruce Croft, Alistair Moffat, C. J. van Rijsbergen, Ross Wilkinson, and Justin Zobel (Eds.). ACM, 335–336. doi:10.1145/290941.291025
- [5] Rocio L. Cecchini, Carlos M. Lorenzetti, Ana G. Maguitman, and Ignacio Ponzoni. 2018. Topic relevance and diversity in information retrieval from large datasets: A multi-objective evolutionary algorithm approach. *Applied Soft Computing* 69 (2018), 749–770.
- [6] Charles L. A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. 2008. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*. ACM, 659–666. doi:10.1145/1390334.1390446
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv abs/1810.04805* (2019).
- [8] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2023. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. *Proc. VLDB Endow.* 16, 10 (2023), 2458 – 2470. doi:10.14778/3603581.3603587
- [9] Marina Drosou and Evaggelia Pitoura. 2015. Multiple Radii DisC Diversity: Result Diversification Based on Dissimilarity and Coverage. *ACM Trans. Database Syst.* 40, 1 (2015), 4:1–4:43. doi:10.1145/2699499
- [10] 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. doi:10.1145/3555041.3589409
- [11] 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. *PVLDB* 16, 7 (2023), 1726–1739.
- [12] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds Mach.* 30, 4 (2020), 681–694.
- [13] Xavier Franch, Andreas Jedlitschka, and Silverio Martínez-Fernández. 2023. A Requirements Engineering Perspective to AI-Based Systems Development: A Vision Paper. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 223–232. doi:doi.org/10.1007/978-3-031-29786-1\_15
- [14] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. Metam: Goal-Oriented Data Discovery. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2780–2793. doi:10.1109/ICDE5515.2023.00213
- [15] Ruoyuan Gao and Chirag Shah. 2020. Toward creating a fairer ranking in search engine results. *Information Processing & Management* 57, 1 (2020), 102138.
- [16] Sreenivas Gollapudi and Aneesh Sharma. 2009. An axiomatic approach for result diversification. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl (Eds.). ACM, 381–390. doi:10.1145/1526709.1526761
- [17] Yue Gong, Zhiru Zhu, Sainyam Galhotra, and Raul Castro Fernandez. 2023. Ver: View Discovery in the Wild. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 503–516. doi:10.1109/ICDE5515.2023.00045
- [18] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. 2023. Data Lakes: A Survey of Functions and Systems. *IEEE Trans. Knowl. Data Eng.* 35, 12 (2023), 12571–12590. doi:10.1109/TKDE.2023.3270101
- [19] Minbiao Han, Jonathan Light, Steven Xia, Sainyam Galhotra, Raul Castro Fernandez, and Haifeng Xu. 2023. A Data-Centric Online Market for Machine Learning: From Discovery to Pricing. *CoRR abs/2310.17843* (2023). doi:10.48550/ARXIV.2310.17843 arXiv:2310.17843
- [20] 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, Toronto, Canada, July 9-14, 2023*. Association for Computational Linguistics, 3786–3800. <https://aclanthology.org/2023.findings-acl.233>
- [21] IMDb. 2022. <https://datasets.imdbws.com/>
- [22] Anoop Jain, Parag Sarda, and Jayant R. Haritsa. 2003. Providing Diversity in K-Nearest Neighbor Query Results. *CoRR cs.DB/0310028* (2003). <http://arxiv.org/abs/cs/0310028>
- [23] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).
- [24] Aamod Khatiwada, Grace Fan, Roe Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1 (2023), Article 9. doi:10.1145/3588689
- [25] Aamod Khatiwada, Harsha Kokel, Ibrahim Abdelaziz, Subhajt Chaudhury, Julian Dolby, Oktie Hassanzadeh, Zhenhan Huang, Tejaswini Pedapati, Horst Samulowitz, and Kavitha Srinivas. 2025. TabSketchFM: Sketch-Based Tabular Representation Learning for Data Discovery over Data Lakes. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1523–1536. doi:10.1109/ICDE65448.2025.00118
- [26] Aamod Khatiwada, Roe Shraga, Wolfgang Gatterbauer, and Renée J. Miller. 2022. Integrating Data Lake Tables. *Proc. VLDB Endow.* 16, 4 (2022), 932–945. doi:10.14778/3574245.3574274
- [27] Aamod Khatiwada, Roe Shraga, and Renée J. Miller. 2023. DIALITE: Discover, Align and Integrate Open Data Tables. In *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*. ACM, 187–190. doi:10.1145/3555041.3589732
- [28] Aamod Khatiwada, Roe Shraga, and Renée J. Miller. 2025. Diverse Unionable Tuple Search: Novelty-Driven Discovery in Data Lakes [Technical Report]. [https://github.com/northeastern-datalab/dust/blob/main/dust\\_technical\\_report.pdf](https://github.com/northeastern-datalab/dust/blob/main/dust_technical_report.pdf)
- [29] Ketii Korini and Christian Bizer. 2023. Column Type Annotation using ChatGPT. In *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023 (CEUR Workshop Proceedings, Vol. 3462)*. CEUR-WS.org. <https://ceur-ws.org/Vol-3462/TADA1.pdf>
- [30] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60. doi:10.14778/3421424.3421431
- [31] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR abs/1907.11692* (2019). arXiv:1907.11692 <http://arxiv.org/abs/1907.11692>
- [32] Hamed Mirzaei and Davood Rafiei. 2023. Table Union Search with Preferences. In *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023 (CEUR Workshop Proceedings, Vol. 3462)*. CEUR-WS.org. <https://ceur-ws.org/Vol-3462/TADA2.pdf>
- [33] Zafeiria Mounoulidou, Andrew McGregor, and Alexandra Meliou. 2021. Diverse Data Selection under Fairness Constraints. In *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus (LIPIcs, Vol. 186)*, Ke Yi and Zhewei Wei (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:25. doi:10.4230/LIPICS.ICDT.2021.13
- [34] Avaniika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.* 16, 4 (dec 2022), 738–746. doi:10.14778/3574245.3574258
- [35] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2022. Responsible Data Integration: Next-generation Challenges. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 2458–2464. doi:10.1145/3514221.3522567
- [36] 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. doi:10.14778/3352063.3352116
- [37] 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. doi:10.14778/3192965.3192973
- [38] OpenAI. 2023. <https://openai.com/chatgpt>
- [39] Koyena Pal, Aamod Khatiwada, Roe Shraga, and Renée J. Miller. 2023. Generative Benchmark Creation for Table Union Search. *CoRR abs/2308.03883* (2023). doi:10.48550/arXiv.2308.03883 arXiv:2308.03883
- [40] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). ACL, 1532–1543. doi:10.3115/V1/D14-1162
- [41] Erhard Rahm and Eric Peukert. 2019. Holistic Schema Matching. In *Encyclopedia of Big Data Technologies*. Springer. doi:10.1007/978-3-319-63962-8\_12-1
- [42] Juan Ramos et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. Citeseer, 29–48.
- [43] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 3980–3990. doi:10.18653/V1/D19-1410
- [44] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. doi:10.1016/0377-0427(87)90125-7
- [45] Roe Shraga and Renée J. Miller. 2023. Explaining Dataset Changes for Semantic Data Versioning with Explain-Da-V. *Proc. VLDB Endow.* 16, 6 (2023),

- 1587–1600. doi:10.14778/3583140.3583169
- [46] Abir Smiti. 2020. A critical overview of outlier detection methods. *Comput. Sci. Rev.* 38 (2020), 100306. doi:10.1016/J.COSREV.2020.100306
- [47] Julia Stoyanovich, Ke Yang, and H. V. Jagadish. 2018. Online Set Selection with Fairness and Diversity Constraints. In *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*. OpenProceedings.org, 241–252. doi:10.5441/002/EDBT.2018.22
- [48] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-trained Language Models. In *SIGMOD Conference 2022*. ACM, 1493–1503. doi:10.1145/3514221.3517906
- [49] Reinier H. van Leuken, Lluís Garcia Pueyo, Ximena Olivares, and Roelof van Zwol. 2009. Visual diversification of image search results. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl (Eds.). ACM, 341–350. doi:10.1145/1526709.1526756
- [50] Saúl Vargas. 2014. Novelty and diversity enhancement and evaluation in recommender systems and information retrieval. In *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast, QLD, Australia - July 06 - 11, 2014*, Shlomo Geva, Andrew Trotman, Peter Bruza, Charles L. A. Clarke, and Kalervo Järvelin (Eds.). ACM, 1281. doi:10.1145/2600428.2610382
- [51] Marcos R. Vieira, Humberto Luiz Razente, Maria Camila Nardini Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina Jr., and Vassilis J. Tsotras. 2011. DivDB: A System for Diversifying Query Results. *Proc. VLDB Endow.* 4, 12 (2011), 1395–1398. <http://www.vldb.org/pvldb/vol4/p1395-vieira.pdf>
- [52] Yunjie Xu and Hainan Yin. 2008. Novelty and topicality in interactive information retrieval. *J. Assoc. Inf. Sci. Technol.* 59, 2 (2008), 201–215. doi:10.1002/ASL.20709
- [53] Ke Yang, Vasilis Gkatzelis, and Julia Stoyanovich. 2019. Balanced Ranking with Diversity Constraints. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 6035–6042. doi:10.24963/IJCAI.2019/836
- [54] Cong Yu, Laks V. S. Lakshmanan, and Sihem Amer-Yahia. 2009. It takes variety to make a world: diversification in recommender systems. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings (ACM International Conference Proceeding Series, Vol. 360)*, Martin L. Kersten, Boris Novikov, Jens Teubner, Vladimir Polutin, and Stefan Manegold (Eds.). ACM, 368–378. doi:10.1145/1516360.1516404
- [55] Kaiping Zheng, Hongzhi Wang, Zhixin Qi, Jianzhong Li, and Hong Gao. 2017. A survey of query result diversification. *Knowl. Inf. Syst.* 51, 1 (2017), 1–36. doi:10.1007/S10115-016-0990-4
- [56] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian J. McAuley, Ke Xu, and Furu Wei. 2020. BERT Loses Patience: Fast and Robust Inference with Early Exit. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. <https://proceedings.neurips.cc/paper/2020/hash/d4dd111a4fd973394238aca5c05bebe3-Abstract.html>
- [57] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD Conference 2019*. ACM, 847–864. doi:10.1145/3299869.3300065
- [58] 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. doi:10.14778/2994509.2994534