POLYJOIN: Semantic Multi-key Joinable Table Search in Data Lakes

Xuming Hu[1]*, **Chuan Lei**[2], **Xiao Qin**[2], **Asterios Katsifodimos**[2], **Christos Faloutsos**[2], **Huzefa Rangwala**[2]
[1]The Hong Kong University of Science and Technology (Guangzhou),
[2]Amazon Web Services
xuminghu97@gmail.com
{chuanlei,drxqin,akatsifo,faloutso,rhuzefa}@amazon.com

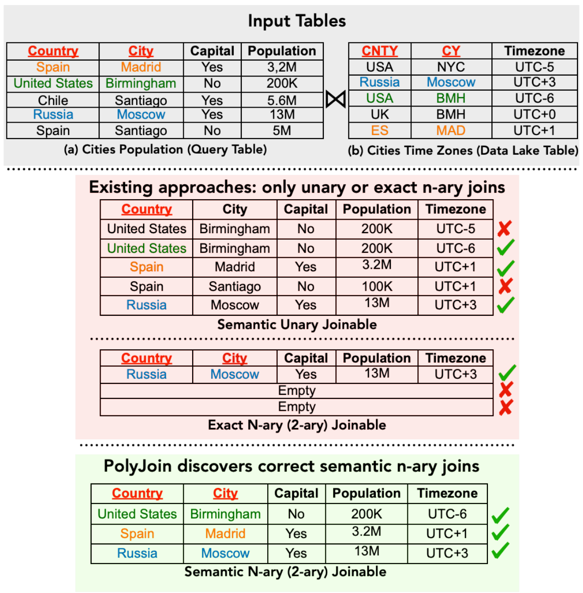
Abstract

Given a query table, how can we effectively discover multi-key joinable tables on the web? Searching and discovering such joinable tables is critical to data analysts and data scientists for reporting, establishing correlations and training machine learning models. Existing joinable table search methods have mostly focused on *single* key (unary) joins, where a single column is the join key. However, these methods are ineffective when dealing with join keys composed of multiple columns (n-ary joins), which are prevalent on web table corpora. In this paper, we introduce POLYJOIN, which finds *multi-key* semantically-joinable tables on the web, given a query table. POLYJOIN employs a multi-key encoder and a novel self-supervised training method to generate the representations of multiple join keys, preserving the alignment across multiple columns. POLYJOIN outperforms the state-of-the-art methods by 2.89% and 3.67% with respect to MAP@30 and R@30 on two real-world benchmarks, respectively.

1 Introduction

Tabular data is one of the most ubiquitous data formats in data lakes. The ability to combine *multiple* tabular datasets enables users to gain insights, identify correlations, and detect patterns that may not be evident when examining individual tabular datasets in isolation (Brickley et al., 2019; Jin et al., 2022; Schlichtkrull et al., 2021). The first step towards combining datasets in a data lake is *joinable table search* (Zhu et al., 2019). Given a table as a query, the objective of joinable table search is to find all tables in a data lake that can be joined with the query table. Joinable table search improves data accessibility and understanding across multiple sources, benefiting tabular data-driven NLP downstream tasks, such as table question answering (Nan et al., 2022; Pal et al., 2023), table fact verification (Zhang et al., 2020; Chen et al., 2019) and table information extraction (Wang et al., 2021a).

*Work done during an internship at Amazon Web Services.


Input Tables
⋈
Existing approaches: only unary or exact n-ary joins
PolyJoin discovers correct semantic n-ary joins

Country	City	Capital	Population
Spain	Madrid	Yes	3,2M
United States	Birmingham	No	200K
Chile	Santiago	Yes	5.6M
Russia	Moscow	Yes	13M
Spain	Santiago	No	5M

(a) Cities Population (Query Table)

CNTY	CY	Timezone
USA	NYC	UTC-5
Russia	Moscow	UTC+3
USA	BMH	UTC-6
UK	BMH	UTC+0
ES	MAD	UTC+1

(b) Cities Time Zones (Data Lake Table)

Country	City	Capital	Population	Timezone	
United States	Birmingham	No	200K	UTC-5	✗
United States	Birmingham	No	200K	UTC-6	✓
Spain	Madrid	Yes	3.2M	UTC+1	✓
Spain	Santiago	No	100K	UTC+1	✗
Russia	Moscow	Yes	13M	UTC+3	✓

Semantic Unary Joinable

Country	City	Capital	Population	Timezone	
Russia	Moscow	Yes	13M	UTC+3	✓
Empty					✗
Empty					✗

Exact N-ary (2-ary) Joinable

Country	City	Capital	Population	Timezone	
United States	Birmingham	No	200K	UTC-6	✓
Spain	Madrid	Yes	3.2M	UTC+1	✓
Russia	Moscow	Yes	13M	UTC+3	✓

Semantic N-ary (2-ary) Joinable

Figure 1: Joinable table search on Open Data.

In Figure 1, the two tables at the top can be joined in two ways: 1) using a *single key join* between the unary key "`Country`" and "`CNTY`" columns in the same figure, or 2) using a *multi-key join* between the n-ary keys ("`Country`", "`City`") and ("`CNTY`", "`CY`") columns. Note that the data instance "`United States`" and "`USA`" cannot form an *exact* join result due to a string mismatch. However, they are semantically equivalent and could be joined with a *semantic join*.

As seen in Table 1, although exact joinable table search has been tackled in the recent years with JOSIE (Zhu et al., 2019) (exact unary joins) and MATE (Esmailoghli et al., 2022) (exact multi-key joins), it becomes increasingly important to leverage language models (LMs) to discovery semantically joinable tables. Specifically, PEXESO (Dong et al., 2021) and DeepJoin (Dong et al., 2022b) make use of embeddings (e.g., BERT (Devlin et al., 2019)) to capture semantic unary joins between tables. Yet, these methods can produce false pos-

Table 1: Join methods and their applicability in different scenarios.

	Multi-Key Joins	Semantic Joins
JOSIE (Zhu et al., 2019)	✗	✗
PEXESO (Dong et al., 2021)	✗	✓
DeepJoin (Dong et al., 2022b)	✗	✓
MATE (Esmailoghli et al., 2022)	✓	✗
POLYJOIN	✓	✓

itive matches (e.g., repeated entries like "`United States, Birmingham`" and "`Spain, Madrid`").

This paper is the first to address the problem of *semantic multi-key joinable table search*. As seen in Figure 1, these joins are required in order to obtain correct join results in case of inconsistent real world data. Semantic multi-key joins pose additional challenges. First, semantic representations of n-ary join keys need to be effectively encoded to preserve the alignment across these join keys. Secondly, there is a lack of labeled data for training and evaluating semantic multi-key join models (Zhu et al., 2019; Esmailoghli et al., 2022).

Approach. In this paper, we propose POLYJOIN, a novel representation learning method for semantic multi-key joinable table search. Given a query table, POLYJOIN retrieves the top-k tables with the highest semantic n-ary joinable scores from a data lake. POLYJOIN is equipped with a multi-key encoder that generates the embeddings of n-ary join keys using the column content as well as column metadata (Section 3.1). POLYJOIN adopts self-supervision signals within and across tables for the model training, as opposed to relying on ground truth labels (Section 3.2). POLYJOIN adopts hierarchical contrastive learning technique to further enhance POLYJOIN's semantic understanding of n-ary join keys (Section 3.3). To evaluate POLYJOIN, we contribute two benchmarks based on Open Data and Kaggle datasets. The original tables are infused with real-world data inconsistencies, and are used to fabricate multi-key joinable tables alongside the corresponding semantic join labels (Section 4).

POLYJOIN demonstrates clear advantages in applications such as table question answering (QA) and data-to-text generation. In complex table QA, POLYJOIN enables reasoning across semantically joinable tables to generate a more accurate answer (Chen et al., 2020). Similarly, in data-to-text generation, POLYJOIN leverages contextual information from joinable tables to create more insightful textual descriptions. By synthesizing data from multiple semantically connected tables, POLYJOIN provides a richer narrative.

POLYJOIN has the following desirable properties. 1) **General**: POLYJOIN is the first method to tackle the problem of semantic multi-key joinable table search in data lakes, using self-supervised learning without the need for labeled data. 2) **Powerful**: POLYJOIN introduces a hierarchical contrastive learning technique to enhance POLYJOIN's semantic representations of multi-key joins, preserving the alignment across these join keys. 3) **Reproducible**: We construct two evaluation benchmarks with real-world noise, enabling realistic evaluation. 4) **Effective**: POLYJOIN outperforms state-of-the-art baselines on real-world benchmarks by 2.89% and 3.67% in MAP@30 and R@30.

2 Problem Definition

Semantic multi-key joinability. Given a candidate table T_D from a data lake and a query table T_Q, we denote the column set of table T_D as C_D and the column set of table T_Q as C_Q. The join keys of table T_D and T_Q are composed of multiple keys (i.e., n-ary join keys), which we denote as $\mathcal{K}_D \subset C_D$ and $\mathcal{K}_Q \subset C_Q$. Note that only if $|\mathcal{K}_D| = |\mathcal{K}_Q| = n$, the two tables are likely to be semantic n-ary joinable. For table T_D, we can obtain a sub-table by extracting only those keys in $\mathcal{K}_D$, and we denote all rows in this sub-table as $\mathcal{R}_D$, likewise for the row set $\mathcal{R}_Q$. Then the semantic multi-key (n-ary) joinability score $\jmath$ between T_D and T_Q is defined as: $\jmath(T_D, T_Q) = \frac{|\mathcal{R}_{match}|}{|\mathcal{R}_Q|}$, where $\mathcal{R}_{match} = \{x \mid x \in \mathcal{R}_Q \wedge \exists y \in \mathcal{R}_D \text{ s.t. } g_{se}(x, y) = 1\}$ and g_{se} is a function that returns 1 if the two rows x & y are semantically equivalent or returns 0 otherwise.

Semantic multi-key joinable table search. Given a set of data lake tables $\mathcal{T}$, a query table T_Q and a value k, the goal is to return the top-k tables from $\mathcal{T}$ sorted by their joinability score $\jmath$ with T_Q. T_Q and top-k tables have n-ary join keys.

3 POLYJOIN's Approach

As motivated in Section 1, semantic multi-key joinable table search needs to address two key challenges: 1) effectively learn semantic representations of multiple keys together within a table by extracting information at the cell level, in order to determine whether a joinable relation exists between two tables, and 2) multi-key joinable table search is often executed over domain-specific and proprietary data lakes. There is a lack of labeled data. Ideally, a method must be designed to be unsupervised and without any manual annotation.

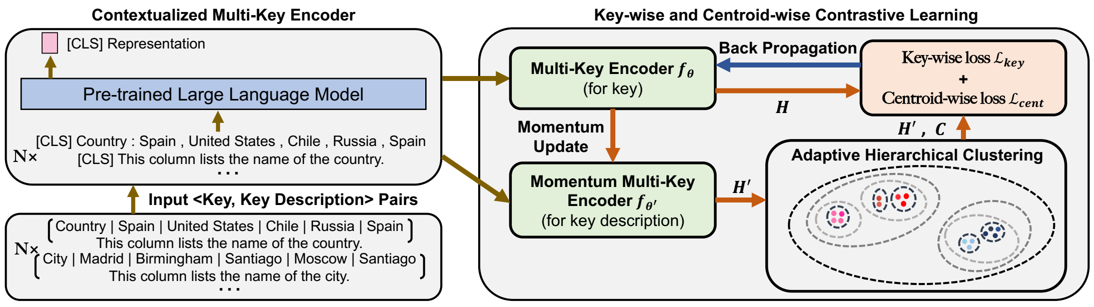

Contextualized Multi-Key Encoder
[CLS] Representation
Pre-trained Large Language Model
[CLS] Country : Spain , United States , Chile , Russia , Spain
[CLS] This column lists the name of the country.
N×
Input <Key, Key Description> Pairs
Country | Spain | United States | Chile | Russia | Spain
This column lists the name of the country.
City | Madrid | Birmingham | Santiago | Moscow | Santiago
This column lists the name of the city.
N×
Key-wise and Centroid-wise Contrastive Learning
Multi-Key Encoder f_θ (for key)
Back Propagation
Key-wise loss $\mathcal{L}_{key}$ + Centroid-wise loss $\mathcal{L}_{cent}$
H
Momentum Update
H', C
Adaptive Hierarchical Clustering
Momentum Multi-Key Encoder $f_{\theta'}$ (for key description)
H'

Figure 2: Overview of PolyJoin.

We propose a representation learning method PolyJoin, which leverages self-supervised signals from column level metadata within and across tables. As shown in Figure 2, we hypothesize that the semantic representations of n-ary keys should align with their descriptions (in metadata). For instance, the description of the column "`Country: Spain, United States, ...`" can be perfectly described by "`This column lists the country names`". Second, multi-key joinable columns across tables tend to describe similar semantic types, such as `Address`, `Person` or `Time`. These columns should have similar representations in the semantic space. Therefore, we employ a hierarchical clustering approach to discover latent semantic clusters and hierarchical contrastive learning to extract self-supervised signals within these clusters. We detail the specific modules in PolyJoin.

3.1 Contextualized Multi-Key Encoder

The contextualized multi-key encoder extracts the column representations from their cell values and obtains semantic representations of multiple columns within a table by concatenating their representations. We leverage pre-trained language model, BERT (Devlin et al., 2019), to effectively encode the semantic representations of columns and their corresponding descriptions. As depicted in Figure 2, we serialize columns $\mathbb{C}$ ($|\mathbb{C}| = N$) with their cells, adhering to the schema proposed in Devlin et al. (2019) and using the reserved $[CLS]$ token to represent the semantic representation of all cells and column descriptions $\mathbb{S}$ in given columns:

$$\begin{aligned}\mathbb{C} &= \big[[CLS], C_1, [SEP], C_2, \cdots, C_N, [SEP]\big],\\ \mathbb{S} &= \big[[CLS], S_1, [SEP], S_2, \cdots, S_N, [SEP]\big],\end{aligned} \tag{1}$$

where C_i denotes the i-th serialized column and S_i denotes the corresponding column description.

We denote the multi-key encoder as $f_\theta(\mathbb{C}, [CLS]_N)$ and $f_{\theta'}(\mathbb{S}, [CLS]_N)$, in which N signifies the number of columns. The encoder produces a fixed-length representation $\mathbf{h}_\mathbb{C} \in \mathbb{R}^{h_R}$ and $\mathbf{h}'_\mathbb{S} \in \mathbb{R}^{h_R}$, where $h_R = 768$ symbolizes the dimension of the $[CLS]$ embedding using BERT. We randomly select N columns from a table and feed them to the multi-key encoder. Consequently, we generate a variety of semantic representations of different column combinations within a table, preserving the alignment across multiple columns.

3.2 Adaptive Hierarchical Clustering

We often observe a hierarchical nature in columns and their descriptions across different tables, as these tables in a data lake are usually organized into hierarchies[1]. For instance, the column descriptions can be 1) the number of years, 2) the number of weeks in a given year and 3) the start date of the reporting week. These descriptions reveal the time hierarchy with different levels: `Year`, `Year/Week` and `Year/Week/Day`. Consequently, we expect that a clustering method can identify hierarchical cluster centroid representatives in a top-down manner. Conventional clustering methods such as k-means (MacQueen, 1967) aggregate data points into a pre-determined number of clusters, but these techniques fail to harness the hierarchical information in a dataset and require preset the number of clusters. Hence, in this paper, we utilize adaptive hierarchical clustering, which has the following advantages: 1) it treats all feature points as potential centroids and extracts underlying hierarchical cluster structures through their mutual similarities and 2) prior knowledge of either the actual number of target clusters or their distributions is not necessary.

The adaptive hierarchical clustering (Frey and Dueck, 2007) exchanges real-valued messages between points until a high-quality set of centroids and their corresponding clusters are generated. The similarity s_{ij} of $\mathbf{h}'_i$ and $\mathbf{h}'_j$ is the distance between these two points and denotes the suitability of j serving as the clustering centroid of i with a_{ij}, and

[1] https://tinyurl.com/bdh6zmy6, https://tinyurl.com/9kcyppdn, https://tinyurl.com/yc65sjc2

indicates the appropriateness of i selecting j as its clustering centroid with b_{ij}:

$$s_{ij} = -\left\|\mathbf{h}'_i - \mathbf{h}'_j\right\|^2, \tag{2}$$

$$a_{ij} = s_{ij} - \max_{j' \neq j}\left(s_{ij'} + b_{ij'}\right), \tag{3}$$

$$b_{ij} = \begin{cases} \sum_{i' \neq i} \max\left(0, a_{i'j}\right), j = i \\ \min\left[0, a_{jj} + \sum\limits_{i' \notin \{i,j\}} \max\left(0, a_{i'j}\right)\right], j \neq i \end{cases} \tag{4}$$

where a_{ij} and b_{ij} are updated through nested iterations until convergence. Then, we aim to find a series of L continuous clustering layers to reveal more fine-grained semantics in the column descriptions. In other words, the points to be clustered at the l-th layer are closer to the corresponding cluster centroid of the layer $l-1$. We use a parameter, $\min(s_{ij}) + \frac{\text{median}(s_{ij}) - \min(s_{ij})}{L-1} \cdot (l-1), l \in [1, L]$, to obtain the l-th layer clustering results, where a larger value leads to a greater number of clusters (Moiane and Machado, 2018).

3.3 Hierarchical Contrastive Learning Loss

Given a table comprising N columns and their corresponding descriptions, the contextualized multi-key encoder generates their semantic representations. We presume that the columns and their descriptions share similar representations in the semantic space, and utilize column-level contrastive learning to extract self-supervised signals:

$$\mathcal{L}_{\text{key}} = \sum_{i=1}^{N} -\log \frac{\exp(\mathbf{h}_i \cdot \mathbf{h}'_i/\tau)}{\sum_{j=1}^{J} \exp(\mathbf{h}_i \cdot \mathbf{h}'_j/\tau)}, \tag{5}$$

where $\mathbf{h}_i$ and $\mathbf{h}'_i$ are the representations of the i-th column and its description, and $\mathbf{h}'_j$ encompasses one positive description and $J-1$ negative descriptions from other columns. τ is a temperature hyper-parameter (Wu et al., 2018).

Furthermore, as described in Section 3.2, different columns could potentially share similar semantics at different levels. Different columns cannot be simply regarded as negative examples through column-level contrastive learning. Hence, we introduce contrastive learning at the level of hierarchical centroids:

$$\mathcal{L}_{\text{cent}} = -\sum_{i=1}^{N} \frac{1}{L} \sum_{l=1}^{L} \log \frac{\exp(\mathbf{h}_i \cdot \mathbf{e}^l_j/\tau)}{\sum_{m}^{c_l} \exp(\mathbf{h}_i \cdot \mathbf{e}^l_m/\tau)}, \tag{6}$$

where $j \in [1, c_l]$, $\mathbf{e}^l_j$ is the centroid corresponding to the j-th column at the l-th level, and m indicates all the centroids from 1 to c_l at the l-th level. Given that we have explicitly constrained $\mathbf{h}_i$ and $\mathbf{e}^l_j$ to the approximate representation space, the temperature hyper-parameter τ adopted in Eq. 5 can be shared here. Note that, during the training process, the column's representation $\mathbf{h}_i$ will be updated in each batch, but all the centroids will only be updated after the entire epoch is over. In essence, our objective is referred to as Hierarchical Contrastive Learning Loss $\mathcal{L}_{\text{Overall}}$:

$$\mathcal{L}_{\text{Overall}} = \alpha \cdot \mathcal{L}_{\text{key}} + \beta \cdot \mathcal{L}_{\text{cent}}, \tag{7}$$

where α and β represent the weights of the two losses. For simplicity, we set $\alpha = \beta = 1$. To maintain centroid invariance and to prevent representation offset problems in column descriptions during the period, we need to smoothly update the multi-key encoder's parameters to ensure a relatively stable column description space (He et al., 2020). In practice, we have constructed two encoders: f_θ and $f_{\theta'}$, both are instances of the multi-key encoder. f_θ is used to obtain the column representation and $f_{\theta'}$ is used to obtain the representation of the column description. θ is updated through $\mathcal{L}_{\text{Overall}}$, while θ' is the moving average of the updated θ'.

After we update the encoder f_θ using $\mathcal{L}_{\text{Overall}}$, the momentum encoder $f_{\theta'}$ can advance each epoch in the following way:

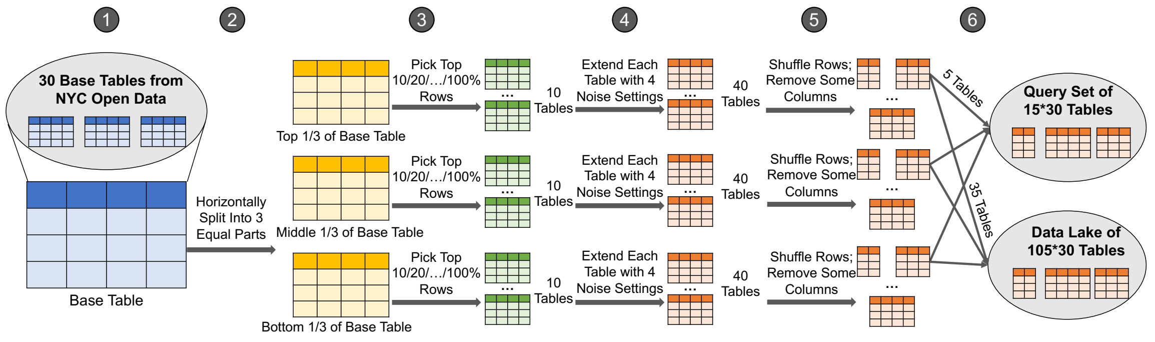
$$\theta' \leftarrow z \cdot \theta' + (1-z) \cdot \theta, \tag{8}$$

where $z \in [0, 1)$ is a coefficient. The momentum update allows θ' to evolve more smoothly than θ, particularly when z approaches 1.

Once the multi-key encoder f_θ is trained, we use it to calculate the semantic joinability scores between a query table T_Q and a data lake table T_D. Following the definition in Section 2, we adopt multi-key encoder f_θ to encode each column in $\mathcal{K}_Q$ and $\mathcal{K}_D$. Given a row $x \in \mathcal{R}_Q$ and $y \in \mathcal{R}_D$, we obtain the embedding of each cell in x and y using f_θ, and then concatenate these embeddings to obtain the row embeddings $\mathbf{h}_x$ and $\mathbf{h}_y$. To calculate the semantic multi-key joinability score $\jmath$, we define the function g_{se} as $g_{se}(x, y) = \mathbb{I}(Cosine(\mathbf{h}_x, \mathbf{h}_y) > \alpha)$, where $\mathbb{I}$ is the indicator function and α is a threshold hyper-parameter ($\alpha = 0.5$ in practice) used in the cosine similarity.

4 Evaluation Benchmark Construction

One critical challenge in evaluating multi-key joinable table search methods is the lack of openly



30 Base Tables from NYC Open Data
Base Table
Horizontally Split Into 3 Equal Parts
Top 1/3 of Base Table
Middle 1/3 of Base Table
Bottom 1/3 of Base Table
Pick Top 10/20/.../100% Rows
10 Tables
Extend Each Table with 4 Noise Settings
40 Tables
Shuffle Rows; Remove Some Columns
5 Tables
35 Tables
Query Set of 15*30 Tables
Data Lake of 105*30 Tables

Figure 3: The construction process of semantic multi-key joinable table search evaluation benchmarks.

available datasets with joinable table ground truth. Existing benchmarks (Esmailoghli et al., 2022) suffer from either lack of multi-key joinable tables or inadequate representation of real-world challenges such as noisy and missing data. In this paper, we create two benchmarks for semantic multi-key joinable table search. Following previous works (Dong et al., 2021; Esmailoghli et al., 2022), we use NYC Open Data[2] and Kaggle datasets[3] to construct these benchmarks, consisting of query tables and the corresponding gold semantic n-ary joinable tables. Figure 3 shows the construction process.

① **Base table selection.** We carefully choose the base tables from both NYC Open Data and Kaggle datasets to generate both query and data lake tables. For NYC Open Data, we first rank all tables based on their *popularity* and select top-30 base tables with 1) more than 4000 rows, 2) more than 5 text columns and 3) more than 5 distinctive column names. These tables ensure that the resulting benchmark contains large tables with a large number of joinable candidates. For Kaggle dataset, we leverage 14 base tables used in MATE (Esmailoghli et al., 2022). For each base table, the columns with the highest number of distinct values are chosen as the n-ary join keys (Dong et al., 2021). These tables also guarantee a large number of joinable candidates in the resulting benchmark.

② **Table horizontal split.** Each base table is horizontally split into three partitions with equal number of rows, yielding three new tables: `table_top`, `table_middle` and `table_bottom`. Consequently, the resulting tables have identical schema but do not share many instances.

③ **Random sub-table generation.** For each `table_top/middle/bottom`, we generate 10 sub-tables by randomly selecting 10/20/.../100% of the rows, creating joins among these sub-tables.

[2]https://opendata.cityofnewyork.us/
[3]https://www.kaggle.com/datasets

Rows (Table)			Columns (Table)			Words (Cell)		
MAX	MIN	Avg.	MAX	MIN	Avg.	MAX	MIN	Avg.
Open Data based Evaluation Benchmark								
50	5	27.5	49	7	17.6	170	1	2.7
Kaggle dataset based Evaluation Benchmark								
50	2	26.8	82	2	19.4	244	1	1.6

Table 2: The statistics of the evaluation benchmarks.

④ **Table augmentation with noise.** We further augment each of the 10 sub-tables into 4 tables with varying real-world noise settings: 1) no noise in column names and values, 2) noise in column names, 3) noise in column values and 4) noise in both column names and values. Therefore, all 40 tables generated from `table_top/middle/bottom` contain real-world noise. We present a detailed noise description in Appendix A.

⑤ **Row shuffling and column removal.** To make the multi-key joinable table search more challenging, we shuffle the rows in a table and randomly remove certain non-key columns as well. This also guarantees joinability among the 40 tables within each `table_top/middle/bottom` source and non-joinability across different sources.

⑥ **Data lake and query tables creation.** Out of the 120 fabricated tables per base table, 15 tables (5 from each `table_top/middle/bottom`) are randomly selected as query tables. The remaining 105 tables constitute the data lake tables. Each query table will have 35 gold semantic joinable tables. Hence, the NYC Open Data benchmark consists of 3150 data lake tables and 450 query tables, respectively. The Kaggle benchmark consists of 1610 data lake tables and 70 query tables, respectively.

To ensure a comprehensive assessment of semantic multi-key joinable table search, we adopted the setting from (Esmailoghli et al., 2022), in which the number of join keys N does not exceed 4. Hence we set $N = 2, 3, 4$ to obtain the corresponding evaluation benchmarks by repeating steps ② – ⑤. Even though the gold multi-key joinable tables are automatically generated during the benchmark construction process, we also manually confirm potential false negatives in the test set. We provide the detailed statistics of the evaluation benchmarks in Table 2. Using these two benchmarks, we evaluate POLYJOIN's multi-key joinable table search capacity and provide a robust measure of POLYJOIN.

5 Experimental Evaluation

5.1 Experimental Setups

Training dataset. We randomly select 3,000 base tables from NYC Open Data as our training set. Note that these tables are not used as part of our experimental evaluation. The training set does not contain label information for semantic multi-key joinable tables. Among these 3,000 tables, a total of 18,824 columns have descriptions, with an average of 12.6 words per column description. First we randomly select N columns from each tables with N varying from 2 to 4. Then we combine them as the inputs to the multi-key encoder to obtain the representations of multiple columns and the ones of their descriptions.

Experimental protocol and hyper-parameter settings. We initialized the multi-key encoder with BERT-Base, BERT-Large (Devlin et al., 2019) and RoBERTa-Large (Liu et al., 2019). We used AdamW (Loshchilov and Hutter, 2017) to optimize the loss. The encoder was trained for 20 epochs with a learning rate of $1e{-}5$. During the adaptive hierarchical clustering stage, we set the numbers of layers L to 3 and the maximum number of iterations to find a clustering centroid to 200, ensuring that the algorithm can stop in time. We set the temperature $\tau = 0.05$ and the momentum parameter $z = 0.9$. We adopted $J = 512$ to balance the quantity of negative samples and the performance of the hierarchical clustering loss optimization.

Evaluation metrics. For evaluation metrics, following previous works (Bogatu et al., 2020; Dong et al., 2022b), we use the Mean Average Precision at K (MAP@K) and Recall at K (R@K) to evaluate the effectiveness of the top-K multi-key joinable table search results. Please note that MAP@K is the average value of Precision at k (P@k), where $k = 1, 2, \cdots, K$. We adopt the $K = 30$. More metric details are provided in Appendix B.

Baseline methods. We compare POLYJOINwith two class of approaches. The first class of methods are designed for joinable table search. Although they do not specifically target semantic multi-key joinable table search, these models can still produce embeddings for single-key joins. We employ the semantic multi-key joinability score defined in Section 2 based on the generated embeddings and evaluate the tables that appear in all joinable results (Esmailoghli et al., 2022). The methods in this first category include **PEXESO⁺** (Dong et al., 2021), **MATE** (Esmailoghli et al., 2022), and **DeepJoin⁺** (Dong et al., 2022b)[4].

The second class of methods use pre-training on the semantics of table cells, thereby obtaining improved semantic column representations. Here, we include **TaBERT** (Yin et al., 2020), **TABBIE** (Iida et al., 2021), **TUTA** (Wang et al., 2021b), **TURL** (Deng et al., 2022) and **TableFormer** (Yang et al., 2022). For a detailed description of these baseline models, please refer to the Appendix C.

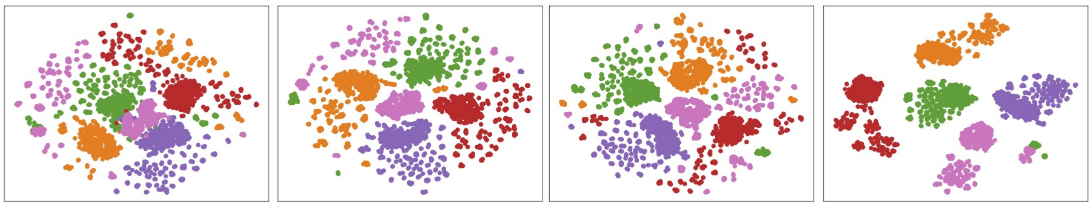
5.2 Results and Analysis

Overall performance. Table 3 shows results for 2-4 multi-key joinable table search over two benchmarks. Our proposed POLYJOIN method consistently surpasses all baseline models in both MAP@30 and R@30, with a significant difference confirmed by the Student's T-test ($p < 0.05$). Specifically, against the formerly top-performing baseline model, TableFormer, POLYJOIN gains a 2.89% and 3.67% improvement in MAP@30 and R@30 respectively. Despite our adjustments to the baseline models for joinable table search to address semantic multi-key joins, they still underperform, notably below other baseline models pretrained on tabular datasets. This holds especially true for the exact n-ary joinable baseline model, MATE, which performs 19.48% worse than POLYJOIN on average. This suggests that in data lakes with unreliable and noisy data, exact-match search isn't robust to semantic discrepancies like acronyms, abbreviations, typos and colloquial terms, leading to a substantial drop in performance. An interesting observation is that employing a larger parameter-based multi-key encoder, like BERT-Large (Devlin et al., 2019) and RoBERTa-Large (Liu et al., 2019), POLYJOIN leads to further performance boosts. For instance, POLYJOIN$_{\text{BERT-Large}}$ and POLYJOIN$_{\text{RoBERTa-Large}}$ sees average improvements of 0.94% and 1.17% in MAP@30 and R@30 across all benchmark settings, respectively.

[4]The $^{+}$ sign indicates that a method uses our semantic

Methods	NYC Open Data based Benchmark						Kaggle based Benchmark					
	2-ary		3-ary		4-ary		2-ary		3-ary		4-ary	
	MAP@30	R@30	MAP@30	R@30	MAP@30	R@30	MAP@30	R@30	MAP@30	R@30	MAP@30	R@30
PEXESO$^+$	61.42	35.63	56.63	32.18	53.26	27.55	65.37	38.42	58.92	32.55	62.01	34.50
MATE	60.71	34.95	54.93	30.69	46.14	24.89	60.52	34.68	53.05	28.89	55.56	31.30
DeepJoin$^+$	67.10	39.39	67.48	39.16	64.28	36.85	78.18	45.70	78.33	46.36	74.89	42.84
TaBERT	67.62	40.39	68.55	39.58	64.69	36.72	79.03	46.14	78.98	46.95	75.44	43.50
TABBIE	67.76	40.26	68.42	39.63	64.83	37.09	78.44	45.91	78.62	46.70	75.12	43.24
TUTA	67.05	39.82	68.13	38.97	64.25	36.74	78.91	46.08	78.38	46.88	75.35	43.58
TURL	68.32	41.02	69.22	40.03	65.18	37.50	79.76	47.10	79.60	47.38	76.48	44.98
TableFormer	69.04	41.85	70.10	40.89	66.42	38.13	80.57	48.29	81.55	49.03	77.85	47.02
POLYJOIN$_{\text{BERT-Base}}$	**71.13**±0.8	**44.39**±0.6	**71.91**±0.7	**44.01**±0.3	**69.55**±1.0	**42.88**±0.5	**82.59**±0.7	**51.81**±0.4	**85.77**±0.9	**54.49**±0.7	**81.92**±0.7	**49.66**±0.4
w/o n-ary training	68.27±1.6	40.22±2.2	69.18±1.4	39.87±1.8	65.89±1.7	37.92±2.4	79.96±1.8	47.31±1.4	79.75±2.0	47.87±1.6	76.89±1.3	45.11±1.8
w/o $\mathcal{L}_{\text{cent}}$	69.82±0.9	41.56±0.7	70.48±0.6	41.93±0.6	67.03±1.2	39.55±0.8	80.53±1.3	48.92±0.8	81.73±0.7	49.29±0.9	78.87±0.6	46.67±0.7
Re. Description	70.76±0.8	44.10±0.6	71.15±0.7	43.27±0.9	68.87±0.6	42.46±0.7	82.21±0.8	51.08±1.1	85.08±0.5	53.87±0.8	81.08±0.7	48.89±0.7
POLYJOIN$_{\text{BERT-Large}}$	72.47±0.7	45.46±0.5	72.74±0.6	44.83±0.8	70.62±0.8	43.79±0.6	83.51±0.7	52.70±0.9	86.62±0.8	55.23±0.6	82.91±0.7	50.53±0.7
POLYJOIN$_{\text{RoBERTa-Large}}$	72.79±1.2	45.52±0.6	73.20±1.0	45.03±0.7	70.88±1.0	43.91±0.6	83.72±0.9	52.88±1.3	86.90±1.2	55.41±0.9	83.25±1.5	50.69±0.8

Table 3: MAP@30 and R@30 comparisons (%). Results of POLYJOIN are averaged over five runs.



(a) PolyJoin *w/o n-ary training* (b) PolyJoin *w/o* $\mathcal{L}_{\text{cent}}$ (c) TableFormer (d) PolyJoin

Figure 4: POLYJOIN achieves better cluster separation for the multi-key representations on Kaggle benchmark.

Ablation study. We perform ablation tests to ascertain the efficacy of each component in POLYJOIN. We consider several POLYJOIN variants. POLYJOIN *w/o n-ary training* omits 2-4 ary keys' unsupervised training data during the training phase, only taking unary keys' training data into account. This leads to POLYJOIN's loss of self-supervised signals for multi-key alignment. POLYJOIN *w/o* $\mathcal{L}_{\text{cent}}$ discards the centroid-wise loss during training, solely utilizing the key-wise loss, hence neglecting hierarchical clustering signals. As per Table 3, both n-ary training and $\mathcal{L}$cent positively influence POLYJOIN's performance, contributing 4.32% and 2.81% improvements in average of MAP@30 and R@30, respectively. This underscores that unearthing potential hierarchical relations and alignments among multi-keys can equip POLYJOIN with superior contextualized multi-key representations, thus boosting the performance of joinable table search. Moreover, an intriguing experiment involving column description substitution was conducted. Column descriptors were gathered from Open Data and Kaggle datasets. We use ChatGPT to generate column descriptions when they are not provided. The details on using ChatGPT for column description generation can be found in Appendix D. Table 3 illustrates that the usage of ChatGPT-generated column descriptions (POLYJOIN *Re. Description*) delivers results nearly akin to those of POLYJOIN, with a minor disparity of 0.39% in average MAP@30 and R@30. While using a larger base encoder like BERT-Large or RoBERTa-Large naturally boosts performance, our framework's hierarchical clustering component provides complementary gains. Table 3 shows that BERT-Large improves MAP@30 and R@30 by an average of 1.00% and 0.88%, respectively, compared to BERT-Base. This suggests that hierarchical clustering and contrastive learning can enhance the strengths of larger encoders, adding unique advantages to POLYJOIN.

Visualizing multi-key representations. We use t-SNE (Maaten and Hinton, 2008) to visualize the multi-key representation space reduced from $\mathbb{R}^{h_R}$ dimensions. We randomly selected five base tables from the NYC Open Data and obtained the multi-key representations. Figure 4 shows the visualization results of multi-key representations, with their base tables serving as the ground-truth colors. We can see that the POLYJOIN *w/o n-ary training* is unable to assign meaningful semantics to the multi-key representations from different base tables due to the absence of multi-key data in training. The absence of hierarchical contrastive loss makes POLYJOIN *w/o* $\mathcal{L}_{\text{cent}}$ incapable of assigning high confidence to semantic features in different clusters, resulting in a loose distribution. This phenomenon is also in the clustering results where TableFormer

multi-key joinability score.

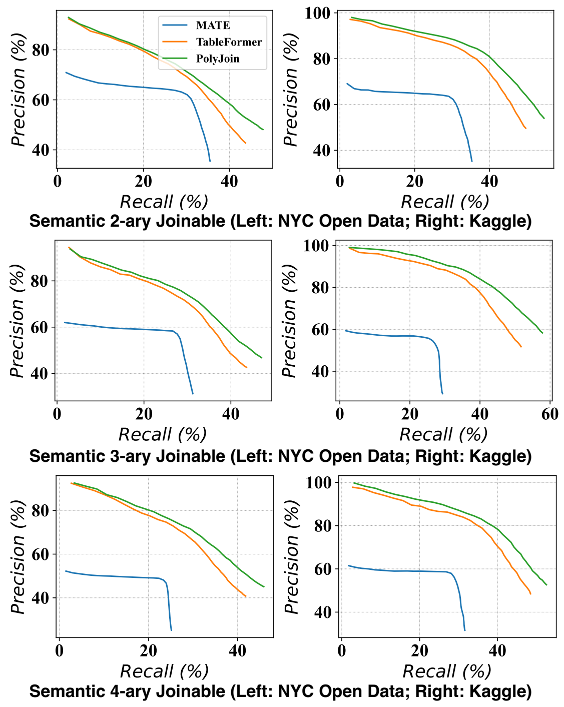


Figure 5: Precision and recall curves on benchmarks.

serves as the multi-key encoder. POLYJOIN effectively utilizes both multi-key data training and multi-key hierarchical structure, thereby achieving dense, clearly delineated cluster results, and a strong semantic representation capability.

Precision vs. Recall. We plot the precision-recall trade-off curve in Figure 5 and observe that as the search threshold decreases, the increase in recall leads to a faster drop in precision. On both benchmarks, POLYJOIN outperforms the baseline model, especially when the threshold decreases; and POLYJOIN can minimize the drop in precision as the recall increases. An interesting finding is that for exact n-ary joinable search baseline, MATE, when the threshold drops, it can hardly find the correct joinable tables in case of real-world settings.

Table 4: Performance comparison on the LakeBench's OpenData and OpenData Large.

Methods	OpenData		OpenData Large	
	MAP@30	R@30	MAP@30	R@30
MATE	31.2%	18.7%	28.8%	16.9 %
TABLEFormer	80.1%	38.2%	79.5%	36.9%
POLYJOIN	**85.2%**	**41.1%**	**84.7%**	**40.7%**

Generalizability of POLYJOIN. We leveraged the LakeBench's OpenData and OpenData Large benchmarks (Deng et al., 2024) with a 3-ary setting to further showcase the generalizability of POLYJOIN. These datasets include diverse and complex real-world data scenarios, providing a robust foundation for assessing the adaptability of joinable table search models in a multi-attribute context. Our results, detailed in Table 4, indicate that POLYJOIN consistently surpasses both MATE and TableFormer on both benchmarks. This confirms POLYJOIN's robustness and generalizability in handling complex data lakes in real-world settings, demonstrating its ability to adapt effectively to unseen and intricate data.

6 Related Work

Joinable table search. Identifying tabular datasets in data lakes is crucial for data scientists (Yu et al., 2020; Cheng et al., 2022; Dong et al., 2022a). Joinable table search is an essential task of tabular data discovery. Existing methods for finding joinable tables focus on either unary joins, ineffective and slow in the existence of n-ary keys (Zhu et al., 2019; Esmailoghli et al., 2022), or syntactic n-ary joins, disregarding substantial real-world noise (Dong et al., 2021, 2022b). Both approaches lead to potential inaccuracies or missed joinable tables. POLYJOIN is an enhanced method for semantic multi-key joinable table search, leveraging multi-key contextual information to boost accuracy.

Self-supervised learning in NLP. Self-supervised learning has been proven effective in NLP applications (Jaiswal et al., 2020; Rani et al., 2023), such as NLU (Hu et al., 2020; Bansal et al., 2020) and QA systems (Banerjee and Baral, 2020; Wilf et al., 2023). This paradigm reduces the dependency on labeled data and saves the time-consuming and labor-intensive data annotation process. Contrastive learning is a commonly used self-supervised learning methods that aims to generate self-supervised signals by pushing semantically similar samples closer in the data representation space (Cui et al., 2020; Huang et al., 2022). POLYJOIN applies contrastive learning to leverage the hierarchical structure within columns to uncover richer multi-key intra-table information.

7 Conclusions

In this paper, we introduce POLYJOIN to effectively identify multi-key joinable tables in data lakes. POLYJOIN uses a multi-key encoder with a self-supervised training to generate representations of n-ary join keys while preserving the alignment across multiple keys. It also uses a hierarchical contrastive learning to enhance its understanding of semantically joinable tables. POLYJOIN outperforms the SOTA by 2.89% and 3.67% in MAP@30 and R@30 on two benchmarks, respectively.

8 Limitations

Our limitations can be analyzed from two different aspects: technical constraints and application-based constraints: 1) On the technical front, our present experiments are limited to BERT-Base, BERT-Large, and RoBERTa-Large as foundational encoders. Adopting larger language models such as LLaMA[5] and Falcon[6] has not been feasible due to the resource constraints at our disposal. 2) In terms of applications, the data employed in our experiments are sourced from NYC Open Data and Kaggle datasets. Beyond the open data, certain domain-specific data, such as those in finance, healthcare and energy, are yet to be incorporated. Additionally, we have exclusively focused on tabular data in English, thus expanding our method to support multi-lingual tabular data is a potential avenue for future research.

References

Pratyay Banerjee and Chitta Baral. 2020. Self-supervised knowledge triplet learning for zero-shot question answering. In *Proc. of EMNLP*, pages 151–162.

Trapit Bansal, Rishikesh Jha, Tsendsuren Munkhdalai, and Andrew McCallum. 2020. Self-supervised meta-learning for few-shot natural language classification tasks. In *Proc. of EMNLP*, pages 522–534.

Alex Bogatu, Alvaro AA Fernandes, Norman W Paton, and Nikolaos Konstantinou. 2020. Dataset discovery in data lakes. In *Proc. of ICDE*, pages 709–720. IEEE.

Dan Brickley, Matthew Burgess, and Natasha Noy. 2019. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *Proc. of WWW*, pages 1365–1375.

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W Cohen. 2020. Open question answering over tables and text. In *International Conference on Learning Representations*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations*.

Zhoujun Cheng, Haoyu Dong, Ran Jia, Pengfei Wu, Shi Han, Fan Cheng, and Dongmei Zhang. 2022. Fortap: Using formulas for numerical-reasoning-aware table pretraining. In *Proc. of ACL*, pages 1150–1166.

[5]https://ai.facebook.com/blog/large-language-model-llama-meta-ai/

[6]https://falconllm.tii.ae/

Wanyun Cui, Guangyu Zheng, and Wei Wang. 2020. Unsupervised natural language inference via decoupled multimodal contrastive learning. In *Proc. of EMNLP*, pages 5511–5520.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, 51(1):33–40.

Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, Kaisen Jin, Chi Zhang, Yuqing Jiang, Yuanfang Zhang, Yuping Wang, Ye Yuan, Guoren Wang, and Nan Tang. 2024. Lakebench: A benchmark for discovering joinable and unionable tables in data lakes. *Proc. VLDB Endow.*, 17(8):1925–1938.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL-HLT*, pages 4171–4186.

Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. 2022a. Table pretraining: A survey on model architectures, pretraining objectives, and downstream tasks. *arXiv preprint arXiv:2201.09745*.

Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *Proc. of ICDE*, pages 456–467. IEEE.

Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2022b. Deepjoin: Joinable table discovery with pre-trained language models. *arXiv preprint arXiv:2212.07588*.

Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2022. Mate: multi-attribute table extraction. *Proceedings of the VLDB Endowment*, 15(8):1684–1696.

Brendan J Frey and Delbert Dueck. 2007. Clustering by passing messages between data points. *science*, 315(5814):972–976.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *Proc. of CVPR*, pages 9729–9738.

Xuming Hu, Lijie Wen, Yusong Xu, Chenwei Zhang, and Philip S. Yu. 2020. Selfore: Self-supervised relational feature learning for open relation extraction. In *Proc. of EMNLP*, pages 3673–3682.

Chao Huang, Lianghao Xia, Xiang Wang, Xiangnan He, and Dawei Yin. 2022. Self-supervised learning for recommendation. In *Proc. of CIKM*, pages 5136–5139.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. Tabbie: Pretrained representations of tabular data. In *Proc. of NAACL-HLT*, pages 3446–3456.

Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. 2020. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2.

Nengzheng Jin, Joanna Siebert, Dongfang Li, and Qingcai Chen. 2022. A survey on table question answering: Recent advances. In *Knowledge Graph and Semantic Computing: Knowledge Graph Empowers the Digital Economy: 7th China Conference, CCKS 2022, Qinhuangdao, China, August 24–27, 2022, Revised Selected Papers*, pages 174–186. Springer.

Christos Koutras, Kyriakos Psarakis, George Siachamis, Andra Ionescu, Marios Fragkoulis, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine in action: matching tabular data at scale. *Proc. of the VLDB Endowment*, 14(12):2871–2874.

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. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

J MacQueen. 1967. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA.

André Fenias Moiane and Álvaro Muriel Lima Machado. 2018. Evaluation of the clustering performance of affinity propagation algorithm considering the influence of preference parameter and damping factor. *Boletim de Ciências Geodésicas*, 24:426–441.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, et al. 2022. Fetaqa: free-form table question answering. *Transactions of ACL*, 10:35–49.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. Multitabqa: Generating tabular answers\\for multi-table question answering. *arXiv preprint arXiv:2305.12820*.

Veenu Rani, Syed Tufael Nabi, Munish Kumar, Ajay Mittal, and Krishan Kumar. 2023. Self-supervised learning: A succinct review. *Archives of Computational Methods in Engineering*, pages 1–15.

Michael Sejr Schlichtkrull, Vladimir Karpukhin, Barlas Oguz, Mike Lewis, Wen-tau Yih, and Sebastian Riedel. 2021. Joint verification and reranking for open fact checking over tables. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6787–6799. Association for Computational Linguistics.

Chenguang Wang, Xiao Liu, Zui Chen, Haoyun Hong, Jie Tang, and Dawn Song. 2021a. Zero-shot information extraction as a unified text-to-triple translation. In *Proc. of EMNLP*, pages 1225–1238.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021b. Tuta: tree-based transformers for generally structured table pretraining. In *Proc. of SIGKDD*, pages 1780–1790.

Alex Wilf, Martin Q Ma, Paul Pu Liang, Amir Zadeh, and Louis-Philippe Morency. 2023. Face-to-face contrastive learning for social intelligence question-answering. In *2023 IEEE 17th International Conference on Automatic Face and Gesture Recognition (FG)*, pages 1–7. IEEE.

Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. 2018. Unsupervised feature learning via non-parametric instance discrimination. In *Proc. of CVPR*, pages 3733–3742.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 2022. Tableformer: Robust transformer modeling for table-text encoding. In *Proc. of ACL*, pages 528–537.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proc. of ACL*, pages 8413–8426.

Jifan Yu, Gan Luo, Tong Xiao, Qingyang Zhong, Yuquan Wang, Wenzheng Feng, Junyi Luo, Chenyu Wang, Lei Hou, Juanzi Li, et al. 2020. Mooccube: a large-scale data repository for nlp applications in moocs. In *Proc. of ACL*, pages 3135–3142.

Hongzhi Zhang, Yingyao Wang, Sirui Wang, Xuezhi Cao, Fuzheng Zhang, and Zhongyuan Wang. 2020. Table fact verification with structure-aware transformer. In *Proc. of EMNLP*, pages 1624–1629.

Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proc. of SIGMOD*, pages 847–864.

A Detailed Real-world Noise for Column Names and Column Values

Inspired by previous work (Koutras et al., 2021), we summarize real-world noise based on column names and column values.

Real-world noise in column values. For text columns, we include the following 4 types of noise.

① Characters can be replaced with those adjacent characters on the keyboard. For instance, the word “`keyboard`” might be changed to “`keybiard`”.

② Adjacent keyboard characters can be randomly added to a string. For example, “`keyboard`” could become “`keeyboard`”.

③ Characters might be randomly removed from a string. As an example, “`keyboard`” might turn into “`keybard`”.

④ Adjacent characters in a string can be incorrectly repeated, changing from “`keyboard`” to “`keyyboard`”.

We also introduce 2 kinds of noise to numeric columns.

① The numeral might be expressed in scientific notation. For example, 15000 could be represented as 1.5e4.

② The numerical value may be altered in a random fashion according to the distribution of values in the column (with changes to mean and variance, followed by random data sampling). As such, 15000 might be changed to 16000.

Real-world noise in column names. We introduce the following 5 types of noise to column names.

① Column names could be attached with its table name. For example, the column “`flight_origin`” from the table “`airline_data`” could be changed to “`airline_data_flight_origin`”.

② Column names could be abbreviated by randomly retaining 25%-50% prefix of each word segment. For instance, “`flight_origin`” might be abbreviated to “`fl_orig`”.

③ Vowels might be dropped from the column names. In this case, “`flight_origin`” could become “`flght_rgn`”.

④ A synonym substitution could occur, if available. For instance, “`flight_origin`” might become “`flight_departure`”.

⑤ Column names might be abbreviated into acronyms. For example, “`flight_origin`” could be shortened to “`fo`”.

B Evaluation Metrics

We utilize the Mean Average Precision at K (MAP@K) and Recall at K (R@K) to gauge the efficacy of the top-K table results retrieved through the search method. Note that MAP@K represents the mean value of Precision at k (P@k), where k ranges from 1 through K. Formally, given a query table Q and a collection of data lake tables $\mathcal{T}$, we denote $\mathcal{T}_Q$ as the set of semantic n-ary joinable tables, based on the known ground truth, and $\mathcal{T}_Q^{'}$ as the set of top-K semantic n-ary joinable tables returned by the searching method. The calculations for P@K and R@K are as follows:

$$P@K = \frac{\mathcal{T}_Q \cap \mathcal{T}_Q^{'}}{\mathcal{T}_Q^{'}}, R@K = \frac{\mathcal{T}_Q \cap \mathcal{T}_Q^{'}}{\mathcal{T}_Q} \qquad (9)$$

Note that perfect R@K is not possible when the ground truth contains less than K joinable tables. We define the Mean Average Precision MAP@K as:

$$MAP@K = \frac{1}{K}\sum_{k=1}^{K} P@k \qquad (10)$$

C Detailed Descriptions of Baseline Models

We compare PolyJoin against base models in two categories. The models in the first category employ a range of column features including column representations to calculate the joinability score between a pair tables.

① PEXESO (Dong et al., 2021) is a framework for discovering unary joinable tables in data lakes. It efficiently finds such tables via a block-and-verify method, utilizing pivot-based filtering and partitioning techniques. It outperforms equi-joins and other similarity-based approaches.

② MATE (Esmailoghli et al., 2022) is a table discovery system utilizing a unique hash-based index for exact n-ary join discovery, employing a space-efficient super key and a filtering layer with Xash, a hash function for efficient table pruning.

③ DeepJoin (Dong et al., 2022b) is an embedding-based retrieval model for unary joinable table discovery. It leverages a pre-trained language model, supports equi-joins and semantic joins, and employs a scalable, log-time nearest neighbor search algorithm.

Both PEXESO$^+$ and DeepJoin$^+$ in Section 5 use the semantic unary join methods from PEXESO and DeepJoin to acquire unary joinable tables, respectively. Moreover, we collect all joinable columns by connecting the representations of multiple keys, and they have also utilized our evaluation method to obtain scores for multi-key joinable tables.

The secondary category employs unlabeled tables for self-guided pre-training, delivering notable outcomes in table comprehension tasks as base encoders for deducing the table joinability score:

④ TaBERT (Yin et al., 2020), a pretrained language model, concurrently learns representations for tables, both fully and semi-structured, and verbal language phrases. TaBERT is trained on a comprehensive corpus comprising 26 million tables and their corresponding English contexts.

⑤ TABBIE (Iida et al., 2021) establishes a simple pretraining target, namely corrupt cell identification, that solely learns from tabular data, setting a benchmark for table-centric tasks. TABBIE, in contrast to other methods, provides embeddings for all table substructures such as cells, rows, and columns. Additionally, it requires significantly less computational power for training.

⑥ TUTA (Wang et al., 2021b) uses a unified pretraining architecture for understanding generally structured tables. TUTA augments transformers with three structure-aware techniques to leverage spatial, hierarchical and semantic information for table understanding.

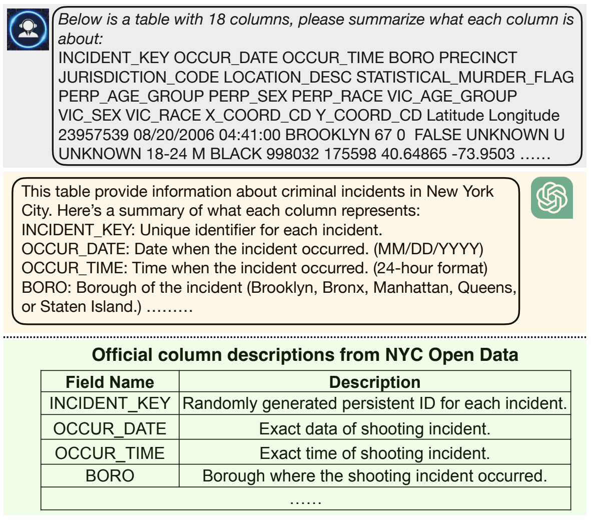
⑦ TURL (Deng et al., 2022) presents a model using a pre-training/finetuning framework on web tables with relational data. It acquires deep contextualized representations, deploys a structure-aware Transformer encoder and uses a Masked Entity Recovery objective during pre-training.

⑧ TableFormer (Yang et al., 2022) introduces a structure-aware table-text encoding architecture where learnable attention biases are used to extract tabular structural biases comprehensively.

Although additional models might be available in this category, we choose the most representative state-of-the-art ones to compare against in this paper.

D LLM-based Column Description Generation

The column descriptions in data lakes might be domain-specific and not always at hand. Thus, we attempt to employ ChatGPT[7] for the generation of column descriptions, enabling PolyJoin to be applied to any other domain-specific tabular data without the need for further manual annotations. As depicted in Figure 6, ChatGPT is capable of producing column descriptions virtually identical to the supplied ones.



Below is a table with 18 columns, please summarize what each column is about:
INCIDENT_KEY OCCUR_DATE OCCUR_TIME BORO PRECINCT JURISDICTION_CODE LOCATION_DESC STATISTICAL_MURDER_FLAG PERP_AGE_GROUP PERP_SEX PERP_RACE VIC_AGE_GROUP VIC_SEX VIC_RACE X_COORD_CD Y_COORD_CD Latitude Longitude
23957539 08/20/2006 04:41:00 BROOKLYN 67 0 FALSE UNKNOWN U UNKNOWN 18-24 M BLACK 998032 175598 40.64865 -73.9503 ……

This table provide information about criminal incidents in New York City. Here's a summary of what each column represents:
INCIDENT_KEY: Unique identifier for each incident.
OCCUR_DATE: Date when the incident occurred. (MM/DD/YYYY)
OCCUR_TIME: Time when the incident occurred. (24-hour format)
BORO: Borough of the incident (Brooklyn, Bronx, Manhattan, Queens, or Staten Island.) ………

Official column descriptions from NYC Open Data

Field Name	Description
INCIDENT_KEY	Randomly generated persistent ID for each incident.
OCCUR_DATE	Exact data of shooting incident.
OCCUR_TIME	Exact time of shooting incident.
BORO	Borough where the shooting incident occurred.
……	

Figure 6: The process of using ChatGPT to automatically generate descriptions for columns.

[7]https://chat.openai.com/