OmniMatch: Joinability Discovery in Data Products

Christos Koutras*
TU Delft
c.koutras@tudelf.nl

Jiani Zhang*
Google
jianizhang@google.com

Xiao Qin
AWS
drxqin@amazon.com

Chuan Lei
AWS
chuanlei@amazon.com

Vasileios Ioannidis
AWS
ivasilei@amazon.com

Christos Faloutsos
AWS & CMU
faloutso@amazon.com

George Karypis
AWS
gkarypis@amazon.com

Asterios Katsifodimos
AWS & TU Delft
a.katsifodimos@tudelft.nl

ABSTRACT

We propose *OmniMatch*, a novel joinability discovery technique, specifically tailored for the needs of *data products*: cohesive curated collections of tabular datasets. *OmniMatch* combines multiple column-pair similarity measures leveraging self-supervised Graph Neural Networks (GNNs). *OmniMatch*'s GNN captures column relatedness by leveraging graph neighborhood information, significantly improving the recall of joinability discovery tasks. At the same time, *OmniMatch* increases its precision by augmenting its training data with negative column join examples through an automated negative example generation process. Compared to the state-of-the-art, *OmniMatch* exhibits up to 14% higher effectiveness in F1 score and AUC without relying on individual, user-provided thresholds for each similarity metric.

PVLDB Reference Format:
Christos Koutras, Jiani Zhang, Xiao Qin, Chuan Lei, Vasileios Ioannidis, Christos Faloutsos, George Karypis, and Asterios Katsifodimos.
OmniMatch: Joinability Discovery in Data Products. PVLDB, 18(11): 4588 - 4601, 2025.
doi:10.14778/3749646.3749715

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/amazon-science/omnimatch.

1 INTRODUCTION

A *data product* is a collection of data assets (e.g., tables) organized under a business context to provide structured, accessible data for specific use cases [14, 42]. It includes metadata, descriptions, governance policies, and access rules to ensure seamless usage. Cloud data management services like Microsoft Purview, Google Dataplex, and AWS DataZone use data products to enable scalable data discovery and governance. Specifically, in the context of tabular data repositories, data products represent cohesive, curated collections of tabular datasets, including database tables, CSV files, and spreadsheets. They may also include metadata such as searchable keywords, auto-generated descriptions, and user-friendly table or attribute names [73].

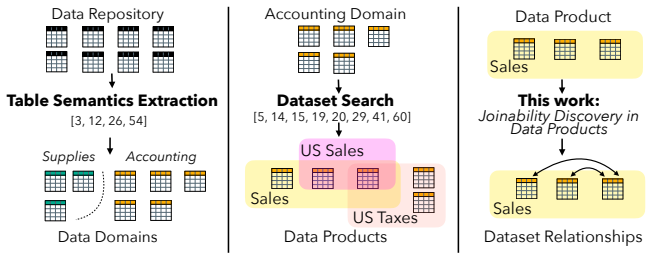


Figure 1: Existing methods can be applied on large data repositories to create coarser (data domains) or finer grained (data products) clusters of datasets. *OmniMatch* takes place inside data products to reveal joinability relationships between column pairs of corresponding datasets.

Prior to data product creation, data processing procedures that facilitate dataset refinement should take place. As shown in the left part of Figure 1, there exists a multitude of *table semantics extraction* methods [4, 16, 35, 72] that can be applied on datasets belonging to a large repository to produce *data domains*, usually by leveraging embeddings summarizing the information stored on a table level. Essentially, data domains are clusters of datasets that contain entities and information belonging to coarser semantic categories (e.g. Supplies and Accounting). To create more fine-grained collections of datasets for specific business needs and use cases, i.e. data products, *dataset search* [6, 18, 19, 23, 24, 38, 54, 80] methods can provide indications on potential dataset-level relationships in an existing data domain. As we see in the middle part of Figure 1, information extracted from search methods can help towards the formation of data products containing datasets that store information with finer grained semantics (e.g. Sales, US Sales and US Taxes).

A fundamental attribute of a well-organized data product is metadata about *joinability* across datasets. Joinability metadata plays a vital role in facilitating the exploration and exploitation of datasets. Data scientists training machine learning (ML) models, can leverage joinability metadata to identify related datasets that

*Work done at AWS.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749715

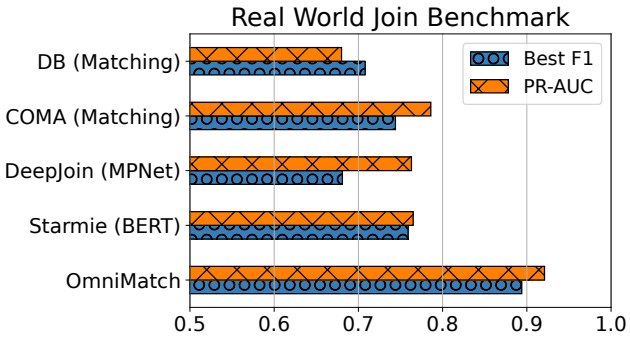


Figure 2: *OmniMatch* outperforms the state-of-the-art column matching and representation methods in terms of best F1 and Precision-Recall AUC scores.

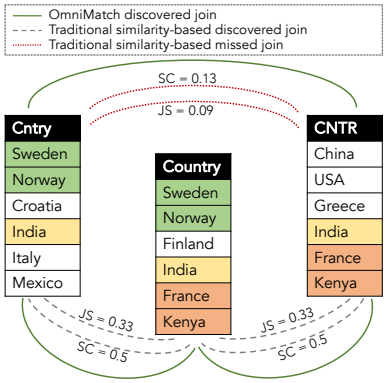


Figure 3: *OmniMatch* at work: (best viewed in color) traditional similarity-based methods vs. *OmniMatch*.

provide additional features, thereby improving the accuracy of an ML model [11, 36, 79]. Joinability metadata can also help in data cleaning by enabling the discovery of new sources of information that serve as ground truth for error checking, inferring missing values, or eliminating duplicates. In addition, joinability metadata is used to automatically generate SQL queries from natural language [40]. Specifically, the joins are retrieved from a catalog, and injected as context to LLMs in order to achieve higher accuracy in NL2SQL translation [64]. Joins can also enable and facilitate table-retrieval [63, 76] and table QA tasks [43]. Finally, automated methods for Tab2Graph [29] require high-quality joinability metadata in order to convert relational tables into graphs automatically and train Graph Neural Networks.

Joinability Discovery in Data Products. In this paper we focus on discovering all joinability relationships across datasets within data products (right part of Figure 1). Unlike dataset search techniques that focus on top-k relationships across datasets with the goal of scaling to large data lakes [6, 13, 18, 23, 24, 33, 38, 52, 54, 80], in this work we focus on finding joins between columns aiming at *high accuracy in smaller numbers of datasets* – even at the expense of increased computation cost. Joinability discovery, in this problem setting, is considered an offline process that can be triggered after a data engineer has already discovered and curated a set of datasets from a large data lake.

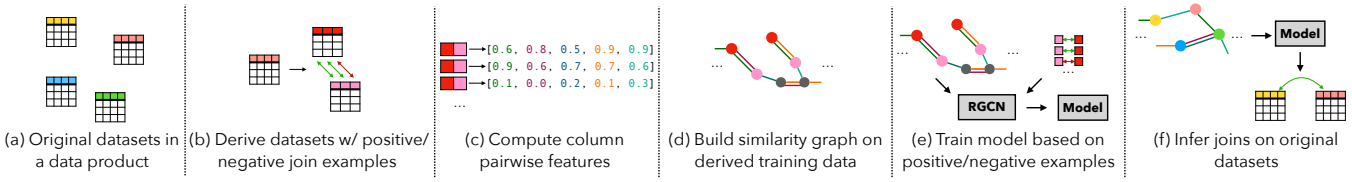
Challenges. A join between two columns requires a non-zero overlap among their value sets. However, it is hard to quantify the usefulness of overlaps: fixed thresholds on set similarity metrics, such as Jaccard Index, might increase false negative/positive rates (due to high/low thresholds respectively). In addition, joins between columns can exist even when their contents differ syntactically. In the literature, those are termed *fuzzy joins* [2, 10, 47, 66, 67]. Fuzzy joins require similarity metrics that capture relatedness beyond value overlaps. This raises the question of which similarity metrics should be used to ensure high effectiveness. Finally, without metadata, such as column names and descriptions, finding joins requires understanding the column contents' semantics.

***OmniMatch*: Effective Joinability Discovery.** In this paper we present *OmniMatch*, a novel self-supervised approach that targets the problem of joinability discovery in data products. As depicted in Figure 2, *OmniMatch* effectively addresses issues associated with existing joinability discovery methods. It does so, in the following ways: *i*) *Enhanced similarity metrics*: *OmniMatch* leverages a diverse suite of similarity metrics between column pairs from different datasets, enabling a more comprehensive understanding of column relatedness. *ii*) *Flexible join detection*: *OmniMatch* considers both equi and fuzzy joins by consolidating and propagating various similarity signals using a variant of *Graph Neural Networks* (GNNs) [61], effectively handling diverse join conditions. *iii*) *Robustness to data noise*: by incorporating a graph-based representation that captures the inherent structure of column relationships, *OmniMatch* can handle noise and perturbations in the input datasets, resulting in more accurate joinability discovery outcomes. *iv*) *Metadata independence*: *OmniMatch* focuses more on the column content data and utilizes the column relatedness information captured in the graph, allowing it to perform join discovery even when metadata is noisy or unavailable. *v*) *Data labeling free*: *OmniMatch* employs a self-supervised learning approach by generating join examples from the original datasets, completely eliminating the need for large amounts of labeled data. This makes *OmniMatch* practical and applicable in data-scarce or labeling-challenged scenarios.

Intuition. Figure 3 depicts three datasets with different similarity scores (Jaccard Similarity – JS and Set Containment – SC). The column pairs (`Country`, `CNTR`) and (`Cntry`, `CNTR`) have high similarities, while `Cntry` and `CNTR` have very low similarities. Traditional similarity-based methods rely on a user-defined threshold, often set at a low value (i.e., JS$\geq$0.09) to discover those joins, negatively affecting precision. In contrast, *OmniMatch* harnesses the power of GNNs with message-passing mechanisms, utilizing graph neighborhood information. This approach allows the discovery of joins that remain undetectable when using threshold-based discovery methods. By leveraging GNNs, *OmniMatch* enhances the precision of join discovery without the need for a predefined threshold.

Contributions. This paper makes the following contributions:

- *OmniMatch* takes a self-supervised approach to find equi and fuzzy joins among tabular datasets in data products. To this end, this paper contributes a method for automatically generating positive and negative examples for self-training, leveraging the



(a) Original datasets in a data product | (b) Derive datasets w/ positive/ negative join examples | (c) Compute column pairwise features | (d) Build similarity graph on derived training data | (e) Train model based on positive/negative examples | (f) Infer joins on original datasets

Figure 4: *OmniMatch*'s steps to predict joins among datasets in a data product.

power of GNNs. As a result, the negative join examples used during training make *OmniMatch* robust against false positives.

- *OmniMatch* is the first to combine multiple similarity signals into a graph to represent relationships among columns of different datasets. As a result, *OmniMatch* decreases the number of false negatives by discovering indirect join relationships leveraging the graph neighborhood information.
- *OmniMatch*'s graph modeling scheme can accommodate an expandable set of similarity signals between column pairs that cover: *i*) semantics via column embeddings extracted from existing deep learning approaches, *ii*) value distributions, as well as *iii*) set similarities.
- On real-world data, *OmniMatch* achieves 14% higher F1 and AUC scores, compared to the state-of-the-art column matching and dataset search methods.

2 THE JOINABILITY DISCOVERY PROBLEM

In this work, we aim at capturing column pairs among datasets in a data product that can be joined, i.e., *joinable* column pairs. Noticeably, we care about capturing all such pairs, without assessing the quality of the corresponding join [28] or ranking them based on some defined joinability score, as done in search tasks [19, 80]. Based on these assumptions, we define column joinability and the problem of joinability discovery in data products as follows.

Definition 2.1 (Joinability). Two columns A and B, with corresponding value sets $\mathcal{A}$ and $\mathcal{B}$, are *joinable* if *i*) there exists a function $h : \mathcal{A} \rightarrow \mathcal{B}$ such that $h(\mathcal{A}) \cap \mathcal{B} \not\equiv \emptyset$ and *ii*) they store values from the same domain, i.e., of the same semantic data type.

Based on the above definition, when $h(\mathcal{A}) \equiv \mathcal{A}$, then the two columns represent an *equi-joinable* pair; in Figure 3 we see cases of such joinable column pairs. However, it is often difficult to strictly define the function $h(\cdot)$ that transforms the values of one column to coincide with the ones of the other column syntactically; examples of such functions might drop, rearrange, or abbreviate tokens, as shown in Figure 5. In these cases, the problem of joinability discovery becomes more challenging.

Joinability Discovery in Data Products. A data product $\mathcal{D}$ consists of a set of n tabular datasets $T_i \in \mathcal{D}, 1 \leq i \leq |\mathcal{D}|$, where each dataset stores a set of columns $C_i, 1 \leq i \leq |\mathcal{D}|$. The problem of *joinability discovery* is to capture all potential joinable pairs among columns belonging to different datasets stored in the data product, i.e., all quadruples of the form (T_i, A, T_j, B), with $i \neq j, A \in C_i, B \in C_j$, joinable$(C_i, C_j)$.

3 APPROACH OVERVIEW

Figure 4 summarizes *OmniMatch*'s steps towards building a prediction model for joinability between columns of tabular datasets in a repository.

– *a,b) Creating training examples*: *OmniMatch* utilizes a dedicated *dataset join-pair generator* for the datasets that reside in a given repository (Figure 4b) to establish the self-supervision. The created positive and negative join examples from individual tables serve as supervisions for training *OmniMatch*'s prediction model to discover joinable relationships across tables.

– *c) Pairwise column feature computation*: At the core of *OmniMatch* we featurize all column pairs among the generated joinable datasets by computing several similarity signals that are widely used in the literature for capturing column relatedness (Figure 4c).

– *d) Column similarity graph construction*: Using the features we calculated earlier, we build a similarity graph where columns are connected with different similarity types of edges, each corresponding to a different feature (Figure 4d). To reduce the noise in graph construction, we propose a filtering strategy.

– *e) Training*: Based on the similarity graph, *OmniMatch* leverages the *Relational Graph Convolutional Network* (RGCN) architecture, a variant of GNNs, in conjunction with the positive and negative join examples from the first step, to train a prediction model for joins (Figure 4e).

– *f) Inference on original datasets*: *OmniMatch* is an inductive model and can adapt to new datasets. Specifically, *OmniMatch* repeats the column pairwise feature computation and similarity graph construction steps for the original testing repository datasets. Applying the prediction model on this similarity graph, we can infer joins among the tabular datasets residing in the repository (Figure 4f).

Why Graph Neural Networks (GNNs). The graph-based data model over the columns creates opportunities for *OmniMatch* to use similarity signals that go beyond the profiles of each column. Specifically, *OmniMatch* constructs a multi-relational [7] graph using columns as nodes[1] and edges representing various types of "relatedness" between the nodes. The fact that an edge connects two columns indicates that they are similar according to a pairwise similarity metric (e.g., Jaccard Index or embedding similarity). However, using different signals to predict joinable relationships is non-trivial in such a graph. GNNs can automatically extract signals from the raw input graph through a message passing mechanism. This mechanism generates representations that aggregate diverse neighboring signals via different relations. Specifically, *OmniMatch* adopts the Relational Graph Convolutional Network (RGCN) model [61], a

[1] In the following, we use the terms columns and nodes, interchangeably.

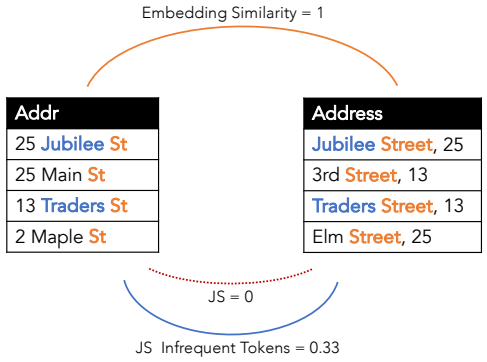


Figure 5: Using Jaccard similarity on infrequent (blue) tokens and embedding similarity on frequent (orange) tokens for capturing fuzzy-joins.

type of GNN that can effectively handle multi-relational data. Intuitively, the joinable relationship discovery can be seen as a learning problem over the constructed multi-relational similarity graph. The RGCN model aims to construct a new graph that consists of the same nodes (columns) but only contains edges that connect the joinable columns. This view is partially observed based on *OmniMatch*'s self-created joinable pairs. Through its learning process, such a partial observation trains the RGCN to gradually learn how to encode signals from a column's profile and its k-hop neighboring columns connected via different relatedness relations (i.e., similarity metrics). Note that *OmniMatch* is inductive and can adapt to unseen datasets.

4 COLUMN SIMILARITIES AS A GRAPH

This section discusses how *OmniMatch* builds a graph representing column relatedness to train a joinability prediction model. We first describe the similarity signals that *OmniMatch* considers. Then, we show how these similarities constitute the basis for building a similarity graph among columns of different tables and analyze the construction process.

4.1 Pairwise Column Similarities

A main part of *OmniMatch* is figuring out how similar two columns are to find possible joins. We picked these similarity signals after many studies on column matching and related dataset search. Next, we explain the set of similarity signals we used in our method and why we use them.

Jaccard Similarity on All Tokens. Jaccard similarity is a widely used similarity metric to assess column relatedness. Specifically, this similarity score is calculated as the size of the intersection divided by the size of the union of the set of values included in two columns (A and B), i.e., $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. Note that for computing this metric, we regard the entirety of the cell values that a column contains, i.e., we consider all tokens. Jaccard similarity is the most commonly used metric to inspect whether two columns store a considerable amount of overlapping values, which is a strong indicator of equi-join relationships [6, 15, 24, 78].

Jaccard Similarity on Infrequent Tokens. Jaccard similarity based on the complete formats of the values stored in columns strongly indicates an equi-join, yet it might be ineffective in fuzzy-joins. This is because even a slight change in the formats of values in one of the columns (e.g., *St* instead of *Street*) might cause the signal to be close to zero. Therefore, it is helpful to include a Jaccard similarity signal based on individual tokens stored in a column rather than on full values. To do so, *OmniMatch* includes a metric used in a state-of-the-art dataset top-k search method [6], which we call Jaccard similarity on infrequent tokens.

Specifically, we first tokenize the values of each column and create a histogram of their occurrences. Then, for each value, with possibly multiple tokens, we keep as its representative the token that has the *lowest frequency*. This enables us to compute Jaccard similarity on the sets of infrequent tokens stored in each column. Intuitively, a high value for this similarity signal indicates a strong relatedness between the corresponding columns since they overlap on tokens that are hardly found in their value sets. Figure 5 depicts an example of a fuzzy join between two columns storing addresses. Using Jaccard similarity on infrequent tokens (i.e., street names), we can capture relatedness between these two columns. On the contrary, Jaccard similarity on full values is zero.

Set Containment. There are multiple cases where Jaccard similarity might be a weak signal of column relatedness, even if the size of overlapping values is relatively large for one of the columns. Essentially, if a column with a small value set is completely covered by another one that stores thousands of discrete values, the Jaccard similarity will be low; indeed, the size of the intersection will be relatively much smaller than the size of the union of values stored in the corresponding columns. To ameliorate this problem, several methods [15, 71] employ *set containment*. Specifically, the set containment from column A to column B is defined as $\frac{|A \cap B|}{|A|}$ and indicates how many unique values from A are included in the intersection with B; a set containment of 1 indicates that those of column B fully cover values of column A. Since this similarity measure is asymmetric, in *OmniMatch*, we choose to include the maximum set containment for a pair of columns (from one to another and vice versa). This way, we include the strongest similarity signal between the two columns. Notably, set containment is significantly effective for capturing *inclusion dependencies* among columns, which is a significant step towards primary key - foreign key (PK-FK) relationship discovery [74].

Embedding Similarity. *OmniMatch* is designed to rely on data instances of the tables, when metadata, such as curated column/table names or descriptions, is not available. Therefore, we compute semantic relatedness for a pairs of columns by using *embedding similarity* of their data instances. Value-based similarity based on pre-trained word embedding models, such as *Glove* [57] and *FastText* [37], has been widely used in related dataset search [6, 18, 39, 54] to capture column semantics of tabular datasets.

In *OmniMatch*, we decided to employ value-based embedding similarity between columns by adopting the approach introduced in [6]. Specifically, for each cell value in a column, we keep the token with the highest occurrence frequency based on the histogram created for computing Jaccard similarity on infrequent tokens. Next, for each such frequent token, we compute a word embedding using FastText, since it can produce representations of any given token, regardless of whether it is included in its vocabulary. The column

representation is then computed as the mean of all embeddings of frequent tokens in the column, and the similarity between the two columns is based on the cosine similarity of their corresponding embeddings. Frequent tokens are usually representative of the column's domain. Hence, basing embedding similarity on them can strongly indicate semantic relevance between columns. For instance, in Figure 5 we see that embedding similarity on frequent tokens (*St* and *Street*) suggests that both columns store values from the same domain (i.e., street addresses).

Distribution Similarity. The last signal that *OmniMatch* considers for a pair of columns is their distribution similarity. Virtually, this type of similarity is often used to capture column relatedness when their value intersection is low (i.e., Jaccard similarity is low) [55, 75], based on the observation that columns storing values from similar domains usually have relevant distributions. Distribution similarity can be beneficial when capturing synonymous terms stored in different columns, which may differ syntactically since we expect them to share similar contexts. Significantly, such a similarity signal could facilitate the discovery of fuzzy joins in *OmniMatch*. Consequently, in *OmniMatch*, we opted for *Jensen-Shanon* (JS) divergence [51] as the distribution similarity measure between two columns, adopting it from [55] where it was found to be effective towards finding similar values for column matching.

Note that *OmniMatch* can be configured to compute other similarity signals due to its flexible design. Essentially, adding similarity signals in the method means adding new types of edges in the similarity graph, as discussed in the following subsection. Therefore, *OmniMatch* can easily be modified to tailor the characteristics of the underlying datasets in a data repository by extending it to include other pairwise column similarities.

4.2 Similarity Graph Construction

OmniMatch's pairwise column similarities can provide strong indicators of join relationships. However, relying solely on a single similarity metric can negatively affect the effectiveness of a joinability discovery method. As we show in Figure 3, a column pair with a low JS score can still be a valid join but will be missed out if a high threshold is chosen. Moreover, value homographs [46] and misleading semantic value matches (e.g., *NY* and *NYC*), suggest that a combination of similarity metrics is a more appropriate choice for the problem of joinability discovery.

Similarity Signals as a Graph. *OmniMatch* uses these similarity signals to construct a *similarity graph*, which encodes important column relatedness information and enables *OmniMatch* to discover indirect join relationships. Specifically, columns from different datasets are transformed into nodes in a graph connected with edges of different types. Each edge type corresponds to a different similarity signal. Such a graph-based data model allows *OmniMatch* to learn *i*) the characteristics of column profiles in join and non-join cases, *ii*) whether different similarity signals contribute to a join or non-join case and *iii*) whether there are graph patterns with pairwise similarity signals and column profiles that constitute a join/non-join case.

Nonetheless, including every type of edge for each pair of nodes would result in a complete graph, which would be difficult to interpret and leverage for joinability discovery. Therefore, we propose two solutions for filtering out unnecessary edges: *i*) based on similarity thresholds, and *ii*) based on top-k similarity ranking.

Edge Selection. The most straightforward approach to filtering out edges would be to choose similarity thresholds for each similarity type. However, if we employ this graph construction technique, we might lose important column relatedness information (and graph connectivity), as it is hard to assess how suitable a value for a threshold is. For example, in Figure 3, using a threshold above `0.5` would filter out all possible edges between the corresponding columns, whereas all column pairs represent a valid join relationship.

To ensure high graph connectivity while accounting for different similarities, *OmniMatch* opts for a different approach: for each node in the graph, it keeps only the top-k edges per node and per type based on the value of the corresponding similarity signal. Essentially, for each node (i.e., column), *OmniMatch* keeps the edges that represent the most prominent join relationships with other nodes. For instance, if we set $k = 1$ in Figure 3, then the only edges that will be kept are the ones between the *Cntry-Country* and *Country-CNTR* pairs. As a result, the edges of the similarity graph that *OmniMatch* constructs using the aforementioned top-k edges represent candidates of potential join relationships between the corresponding columns. Yet, the graph is not guaranteed to contain edges connecting every possible true column join pair (e.g., *Cntry* and *CNTR* share no edges for $k = 1$). As we see in the following section, *OmniMatch* tackles this issue by taking advantage of transitive paths in the similarity graph, to capture joins indirectly.

5 GRAPH MODEL TRAINING

We denote the constructed similarity graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with nodes (columns) $v_i \in \mathcal{V}$ and edges $(v_i, r, v_j) \in \mathcal{E}$, where $r \in \mathcal{R}$ is a relation type indicating one of five similarity relation types defined in Section 4.1. In this section, we discuss how *OmniMatch* leverages the graph $\mathcal{G}$ to learn column representations with GNNs. We begin by exploring the process of creating the initial column features. Subsequently, we employ the message passing paradigm of GNNs to calculate the aggregated column representations and provide a detailed illustration of using RGCNs in *OmniMatch*. Then, we explain how *OmniMatch* automatically creates positive and negative column joins for training and how to use different loss functions to guide training. Finally, we discuss how to do inference.

5.1 Initial Column Features

We describe a column with a collection of identified features that better represent its characteristics [1]. We denote the initial feature vector for a column i as $\mathbf{x}_i \in \mathbb{R}^{d_f}$, where d_f denotes the feature dimension. Specifically, for each column, we use a simple profiler that summarizes statistical information about the values of a given column. We do so since more complex information about the column contents is captured by different types of edges among the nodes in the similarity graph. To this end, we make use of the column profiling component from Sherlock [34] by computing statistics falling into the following two categories:

– *Global statistics.* Those include aggregates on high-level characteristics of a column, e.g., the number of numerical values.

– *Character-level distributions.* For each of the 96 ASCII characters that might be present in the corresponding values of the column, we save character-level distributions.

5.2 Column Representation Learning

Next, we briefly introduce the message-passing architecture of RGCNs, which captures necessary similarity signals within the graph and refines the representation of columns by leveraging neighborhood information.

5.2.1 The message passing paradigm for GNNs. The message passing paradigm follows an iterative scheme of updating node representations based on the aggregation from neighboring nodes. Suppose $\mathbf{h}_i^{(\ell)}$ represents the node representation for column i at iteration ℓ, then the paradigm composes four parts:

(1) Initialization: $\mathbf{h}_i^{(0)} = f_{\theta_1}(\mathbf{x}_i), \forall v_i \in \mathcal{V}$. For each node i, we initialize its node representation $\mathbf{h}_i^{(0)}$ as a function of the feature vector defined in Section 5.1.
(2) Message computation: $\mathbf{m}_{i \leftarrow j}^{(\ell)} = \phi_{\theta_2^{(\ell)}}(\mathbf{h}_j^{(\ell-1)}, \mathbf{h}_i^{(\ell-1)}, \mathbf{e}_{i,j}^{(\ell-1)})$. Function $\phi_{\theta_2^{(\ell)}}(\cdot)$ parameterized by $\theta_2^{(\ell)}$ computes a message from each neighboring node j to the central node i. Here, $\mathbf{e}_{i,j}$ denotes edge information between nodes i and j, which contains the information of a specific relation type.
(3) Neighbor aggregation: $\mathbf{m}_i^{(\ell)} = \psi_{\theta_3^{(\ell)}}(\{\mathbf{m}_{i \leftarrow j}^{(\ell)} | j \in \mathcal{N}_i\})$. This step aggregates the messages received from all the neighboring nodes defined by $\mathcal{N}_i$ to form a comprehensive message for node i. $\psi_{\theta_3^{(\ell)}}(\cdot)$ is the function parameterized by $\theta_3^{(\ell)}$ that aggregates messages.
(4) Message transformation: $\mathbf{h}_i^{(\ell)} = f_{\theta_4^{(\ell)}}(\mathbf{h}_i^{(\ell-1)}, \mathbf{m}_i^{(\ell)})$. Function $f_{\theta_4^{(\ell)}}(\cdot)$ parameterized by $\theta_4^{(\ell)}$ transforms the aggregated information into an updated representation for i.

In summary, the GNN message-passing paradigm initializes node representations, computes messages between neighboring nodes, aggregates these messages, and transforms the aggregated information to update node representations in an iterative manner. A column gains the ability to receive a greater number of relevant messages from its neighbors at the Lth-hop. This enables us to delve into high-order connectivity information and enhance our understanding of intricate relationships within the data. Such high-order connectivities are crucial to encode the similarity signal to estimate the joinability score between two columns. The parameters θ_1, $\theta_2^{(\ell)}$, $\theta_3^{(\ell)}$, and $\theta_4^{(\ell)}$ are adjustable based on different GNN architectures and can be learned during the training of GNN.

5.2.2 Relational Graph Convolutional Networks (RGCNs). In *OmniMatch*, we use the RGCN model [61] to capture multi-relational and multi-hop features effectively. An RGCN explicitly models different relation types, which is critical for handling diverse similarity relationships. The shared weight matrices for each relation reduce overfitting and ensure consistent transformation across relations. Thus, the RGCN model is the right choice to capture nuanced signals of joinability.

Node feature initialization. We set the initial value of $\mathbf{h}_i^{(0)}$ as $\mathbf{x}_i$ in $\mathbb{R}^{d_f}$ defined in Section 5.1, with an empty parameter set θ_1.

Message Computation. Intuitively, the neighboring columns in a similar graph can give more clues about the semantic meaning of a column. We build upon this basis to encourage message feature propagation between linked columns under different types of similarity relations as follows. Relation-specific transformations capture distinct types of similarity, enabling the model to differentiate between various relations. In *OmniMatch*, we use linear transformations as the encoding function:

$$\mathbf{m}_{i \leftarrow j}^{r\ (\ell)} = \frac{1}{|\mathcal{N}_i^r|}(\mathbf{W}_r^{(\ell)}\mathbf{h}_j^{(\ell-1)} + \mathbf{b}_r^{(\ell)}) + \frac{1}{\sum_{r \in \mathcal{R}} |\mathcal{N}_i^r|}(\mathbf{W}_0^{(\ell)}\mathbf{h}_i^{(\ell-1)} + \mathbf{b}_0^{(\ell)}), \tag{1}$$

where $\mathbf{W}_r^{(\ell)} \in \mathbb{R}^{d_h^{(\ell)} \times d_h^{(\ell-1)}}$ is a weight matrix for relation r, which transforms a column feature vector of dimension $d_h^{(\ell-1)}$ to a hidden dimension $d_h^{(\ell)}$. There is also a different weight matrix $\mathbf{W}_0 \in \mathbb{R}^{d_h^{(\ell)} \times d_h^{(\ell-1)}}$ that helps preserve some of the original information (residual connection). So, we have $\theta_2^{(\ell)} = \{\mathbf{W}_r^{(\ell)}, \mathbf{b}_r^{(\ell)}, \mathbf{W}_0^{(\ell)}, \mathbf{b}_0^{(\ell)}\}$. $\mathbf{b}_r^{(\ell)}$ and $\mathbf{b}_0^{(\ell)}$ are the bias vectors. $\mathcal{N}_i^r$ stands for the set of neighboring columns of i under relation $r \in \mathcal{R}$ and $\sum_{r \in \mathcal{R}} |\mathcal{N}_i^r|$ indicates the total number of neighbors under all types of similarities. Thus, the coefficient scalar controls the number of messages being propagated based on the degrees of the node under each relation.

Neighbor Aggregation. In the aggregation stage, messages from neighboring columns are passed to the target column via different types. This helps refine the understanding of our target column i:

$$\mathbf{m}_i^{(\ell)} = \sigma\Big(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \mathbf{m}_{i \leftarrow j}^{r\ (\ell)}\Big), \tag{2}$$

After computing the aggregated specific messages, we sum the messages from all types and pass the output to a non-linear function $\sigma(\cdot)$. Here $\mathbf{m}_i^{(\ell)}$ denotes the representation of column i after aggregating ℓ column propagation layers. We use sigmoid as the activation function $\sigma(\cdot)$, since it allows messages to encode positive signals and filter the negative ones. Summation allows proportional representation of high-degree nodes, and the activation filters out irrelevant signals while retaining critical ones.

Message Transformation. We use the residual connection without any additional parameters to update the node representaion. This residual connection helps preserve original feature information while incorporating learned neighborhood signals. In addition to the messages propagated from the neighbors under different similarity channels, we consider the self-connection of i, which retains the information of the original column features:

$$\mathbf{h}_i^{(\ell)} = \mathbf{h}_i^{(\ell-1)} + \mathbf{m}_i^{(\ell)}. \tag{3}$$

At the L-th layer, the node representations are $\mathbf{h}_i^{(L)}, \forall v_i \in \mathcal{V}$.

5.3 Generating Training Examples

Training our prediction model requires join (positive) and non-join (negative) labels. To do so, *OmniMatch* takes a self-supervised approach, leveraging positive and negative join examples that are automatically generated from the tabular data in the repository. Self-supervised generation of training examples avoids manual labeling

and ensures scalability across large repositories. Specifically, for each table in the input, *OmniMatch* adopts a join pair fabrication process , similar to the ones described in [45, 54, 80]:

- We randomly pick some columns from the input table that the derived pair of datasets will share.
- Then, we split the original dataset's rows into two randomly overlapping sets. Consequently, we create a pair of datasets with a random number of columns and rows.
- To simulate fuzzy-joins, we randomly perturb the data values of one of the two created datasets. We do so only for instances belonging to columns that are shared among the generated tables. To perturb the data values, we either *i*) insert random typos based on keyboard proximity (e.g., *science* becomes *scienxe*) or *ii*) use common alternative values formats for specific column cases (e.g., dates, money amounts, street addresses, etc.).

Based on the above join generation process, we create a pair of joinable tables for each original dataset in the repository (Figure 4b). The columns that join in these pairs are used as positive training examples, while the rest of the column combinations between the two tables are regarded as negative join examples. Note that with this generation process, the derived pairs will share joins of various overlaps and fuzziness, enhancing the robustness of our model.

5.4 Loss Functions

To refine the column representations produced from the RGCN, *OmniMatch* leverages the automatically created positive and negative join examples to train a prediction model. Notably, the choice of loss function can have a considerable impact on the effectiveness of the produced representations. Therefore, in what follows, we describe how to employ two different loss functions towards joinability discovery: *i*) *cross-entropy loss*, which creates a representation space for linearly separating joins from non-joins, and *ii*) *triplet margin loss* which optimizes representations for a given metric distance, to bring joinable columns close and separate non-joinable ones.

Training with cross-entropy loss. The cross-entropy loss encourages the model to assign high similarity scores to positive joinable column pairs and low scores to negative non-joinable pairs. This approach balances positive and negative examples using a weighting factor. In this training procedure, the model's goal is to optimize the following *cross-entropy* loss function:

$$\begin{aligned}\mathcal{L} = &- \sum_{(A,B)\in\mathcal{J}} w_p \cdot \log\sigma(sim(\mathbf{h}_A^{(L)}, \mathbf{h}_B^{(L)})) \\ &- \sum_{(A,B)\in\mathcal{NJ}} \log(1-\sigma(sim(\mathbf{h}_A^{(L)}, \mathbf{h}_B^{(L)}))),\end{aligned} \tag{4}$$

where $\sigma(\cdot)$ is the sigmoid function, while $\mathcal{J}$ and $\mathcal{NJ}$ are the sets of positive and negative column join examples. Notably, the parameter w_p is the weight we use to balance the positive and the negative examples, which we set as the ratio of negative to positive join examples in training. The similarity scores are computed by feeding pairs of RGCN-produced column representations to a *Multilayer Perceptron* (MLP), whose parameters are also learned during training to give correct predictions. With this model training, we aim to compute column representations (using RGCN), so we can build a similarity function (through MLP) that scores join examples higher than non-join ones.

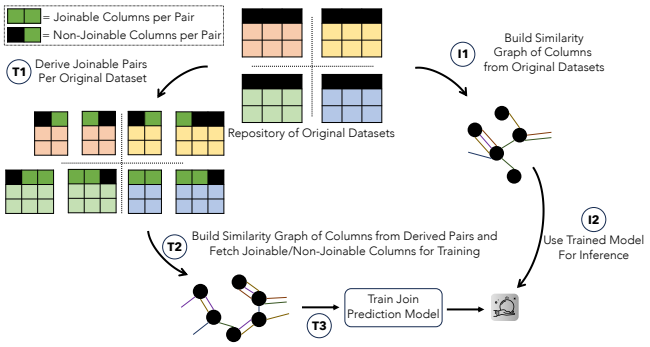


Figure 6: *OmniMatch*'s training and inference procedures.

Training with triplet margin loss. An alternative for proceeding with training is using the triplet margin loss function. The triplet margin loss ensures that positive joinable column pairs are closer in representation space than negative non-joinable pairs, with a margin. The triplet loss explicitly optimizes relative distances, making it robust for ranking-based tasks.

$$\mathcal{L} = \sum_{(A,B^+,C^-)} \max\{d(\mathbf{h}_A^{(L)}, \mathbf{h}_{B^+}^{(L)}) - d(\mathbf{h}_A^{(L)}, \mathbf{h}_{C^-}^{(L)}) + \text{margin},\ 0\} \tag{5}$$

where $d(\cdot,\cdot)$ is a vector distance function, and *margin* is a positive value. For each column, we consider one column that joins (denoted by +) and all others that do not join (denoted by −) based on the generated dataset pairs. Intuitively, training to minimize the triplet margin loss helps the RGCN learn to bring the representations of columns that join, closer than the ones that do not.

5.5 Training and Inference of Join Predictions

Figure 6 summarizes *OmniMatch*'s training and inference procedures. We derive a pair of joinable pairs for each original dataset in the repository, as discussed in Section 5.3. Based on the derived tables, our method first computes all pairwise column similarities and constructs the similarity graph. Then, the joinability prediction model training process is applied to the constructed graph, where learning is guided by one of the two loss functions (Section 5.4).

While model training occurs on the derived dataset pairs, our objective is to discover joins among the columns of the original datasets in the data repository. To this end, *OmniMatch* builds the similarity graph based on the pairwise similarities of columns belonging to the original tabular datasets (right part in Figure 6). Based on the connectivity information of this graph, the trained RGCN model can be directly applied to retrieve the representations of columns: message aggregation takes place once to infer the column embeddings based on the weight matrices learned during training. As a last step, *OmniMatch* uses the column representations to produce a joinability score between each pair of columns coming from different datasets in the repository; the joinability score depends on the loss function used to guide the learning process (Section 5.4).

Benchmark	#Tab.	#Col.	#Equi-Joins	#Fuzzy-Joins
City Government	110	703	1451	128
Culture Recreation	120	687	1254	256
Freyja [52]	58	180	736	124
GDC [49, 60]	10	136	137	157

Table 1: Statistics of the evaluation benchmarks. 'Tab.' stands for 'Table' and 'Col.' stands for 'Column'.

6 EXPERIMENTAL EVALUATION

In this section, we present a comprehensive set of experiments that showcase the effectiveness of *OmniMatch*. First, we describe the joinability discovery benchmarks and baseline methods against which we evaluate our method. Then we provide the experimental results that demonstrate *i*) the gains in effectiveness with respect to state-of-the-art methods when using *OmniMatch*, *ii*) how *OmniMatch*'s prediction model compares to using other models, *iii*) how the model of *OmniMatch* and other self-supervised methods perform when tested on datasets not seen during training, and *iv*) how different similarity signals are related to the model's effectiveness. In addition, we provide execution times for the different steps of our method. We summarize our main results as follows.

- *OmniMatch* is considerably more effective than state-of-the-art column matching and column representation methods.
- We showcase that utilizing only one similarity signal reduces *OmniMatch*'s effectiveness. The degree of reduction depends on the characteristics of the underlying datasets.
- The prediction model of *OmniMatch* is highly effective even when tested on data not seen during training, while it outperforms other self-supervised methods.
- *OmniMatch*'s choice of using RGCNs for leveraging the set of similarity signals is superior to using alternative ML models.

6.1 Experiment Setup

Datasets & Ground Truth. We curate two realistic joinability discovery benchmarks, and make use of another two, recently proposed benchmarks [49, 52]. Table 1 summarizes the statistics of all benchmarks.

We explored the New York City OpenData[2], and created two benchmark data products: *City Government* (11 base tables) and *Culture Recreation* (12 base tables). Within each data product, we manually annotated all the joins between columns across the *base* tables. The captured join pairs have been cross-checked by three annotators who also verified whether there is a value overlap (Def. 2.1) between the ground truth column pairs.

To increase the data volume, and to make the data products more challenging, we derived 10 tables from each base table in the benchmark, using techniques similar to [45, 54], i.e., horizontal/vertical partitions and instance value typos. In particular, for each base table we constructed 5 pairs of joinable tables, sharing a number of columns (1-3) with randomly injected typos, similar to the example generation process in Section 5.3, to simulate fuzzy-joins. Similarly, the Culture Recreation data product consists of 120 tables derived from the 12 base tables. The ground truth of joins for datasets derived from the same base table, is captured automatically [45, 54]. For the datasets derived from different base tables, we make use of the join pairs that we annotated manually as described above. Most columns in both benchmarks store mainly categorical and text data.

In addition, we also conduct experiments on two benchmarks that were introduced in other works, with datasets from different domains. The Freyja benchmark [52] consists of datasets found in open repositories, such as Kaggle and OpenML, for which the authors have manually annotated equi-joins and fuzzy-joins. On the other hand, the GDC benchmark [60] was developed in [49] for matching datasets from tumor analysis studies to the Genomics Data Commons (GDC) schema. To evaluate our method and the other baselines on this benchmark. The ground truth of the GDC benchmark is based on annotations by medical experts.

Measuring Effectiveness. We use *Precision-Recall (PR) curves* to evaluate the effectiveness of *OmniMatch* and the other baseline methods based on the final joinability prediction scores for each column pair among different datasets in the benchmarks. PR curves are suitable for illustrating effectiveness results when there is an imbalanced distribution of labels in the test set. Indeed, in our case, the number of non-joinable column pairs is significantly higher than the number of joinable ones for both benchmarks. A significant advantage of using PR curves is that we can observe effectiveness for varying similarity thresholds, thus making the presentation non-biased. PR curves can help us observe how different similarity thresholds affect a method's performance; stable precision for increasing recall values means that the method's effectiveness is robust to different similarity thresholds. We also report the best F1 and PR-AUC scores to summarize the results shown in PR curves.

State-of-the-art Baselines. We compare *OmniMatch* against the two best-performing column matching methods, according to [45], and the state-of-the-art column representation methods for capturing relatedness among columns, desribed below.

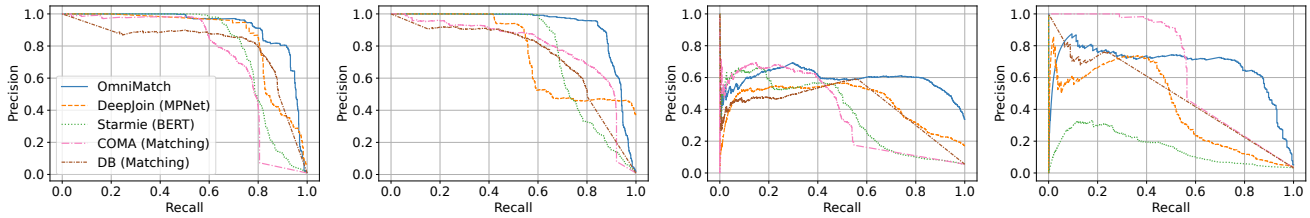
– *COMA* [17] is a seminal matching method that takes into consideration multiple similarity scores, from both metadata and data instances [22]. COMA's effectiveness relies on processing these similarity signals from simple metrics to decide on possible column matches. We make use of the COMA 3.0 Community Edition.

– *Distribution-Based (DB) Matching* [75] is an instance-based column matching method. The method constructs clusters using the *Earth Mover's Distance* (EMD) to capture relatedness among columns of different tabular datasets. During cluster refinement, the method considers exact value overlaps between column pairs to avoid false positives. To include the DB matching method in our experiments, we use the implementation provided by Valentine [45].

– *Starmie* [23] leverages a pre-trained *Language Model* (LM), specifically RoBERTa [48], and contrastive learning [9] to produce column representations towards *unionable dataset search*. In our evaluation, we use Starmie, as shared in a public repository [3], to produce contextualized column representations for the datasets in the input. We then compute the pairwise cosine similarity of the column embeddings among different datasets.

[2]https://opendata.cityofnewyork.us/

[3]https://github.com/megagonlabs/starmie



(a) City Government **(b) Culture Recreation** **(c) Freyja** **(d) GDC**

Figure 7: Effectiveness comparison of *OmniMatch* with other methods for various similarity thresholds.

Best F1 Scores					
Benchmark	***OmniMatch***	**Starmie**	**DeepJoin**	**COMA**	**DB**
City Government	**0.857**	0.781	0.838	0.720	0.803
Culture Recreation	**0.894**	0.759	0.681	0.744	0.708
Freyja	**0.695**	0.509	0.574	0.484	0.576
GDC	**0.735**	0.278	0.541	0.662	0.370
PR-AUC Scores					
Benchmark	***OmniMatch***	**Starmie**	**DeepJoin**	**COMA**	**DB**
City Government	**0.920**	0.798	0.851	0.733	0.760
Culture Recreation	**0.921**	0.765	0.763	0.786	0.680
Freyja	**0.589**	0.370	0.446	0.347	0.305
GDC	**0.679**	0.156	0.414	0.560	0.210

Table 2: Metrics comparison to other methods.

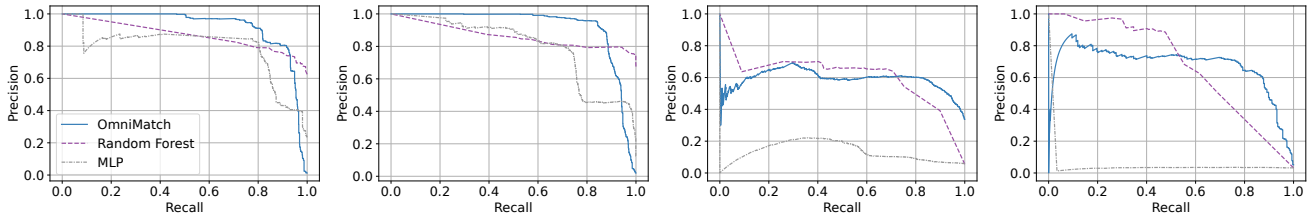
– *DeepJoin* [19] also uses pre-trained LMs, to produce column representations for *joinable table search* in *data lakes*. Specifically, it fine-tunes initial column embeddings from a *sentence transformer* [59], by training a model based on a set of positive join pairs. The latter derives from a similarity join method of choice and a high threshold to ensure lower numbers of false positives. For the needs of our evaluation, we train the DeepJoin model based on the Sentence-BERT[4] library, to produce column representations. Again, we use pairwise cosine similarity as the joinability score between two columns.

Other ML Predictive Methods. We also evaluate the strength of *OmniMatch*'s graph model and RGCN architecture by comparing it to other straightforward column joinability prediction models:
– *Random Forest* considers only the column pairwise similarities and the positive and negative join training examples that our method computes to train a binary classification method, similar to [5]. For our experiments, we use the random forest implementation from `sklearn` [56] Python toolkit, for both training and inference, with `100` decision tree classifiers.
– *MLP* uses the same information as the Random Forest baseline but feeds them to a shallow Multi-Layer Perceptron (MLP) binary classification model (one hidden layer).

Tuning *OmniMatch*. We configure *OmniMatch* by running experiments when varying the model's parameters. By doing so, we came to the following conclusions.
– *Graph Construction:* We trained *OmniMatch*'s joinability prediction model for different values of top-k edges that we consider in the graph for each node and similarity signal to assess changes in effectiveness. Our results showed that using values greater than 5 did not improve our model's effectiveness, hence we proceed with a range search for $1 \leq k \leq 5$ for each of the benchmarks evaluated. In the absence of test-data ground-truth, common heuristic-based strategies can be employed to choose the value of k, such as relying on minimum validation loss during training. We have empirically verified that tuning k based on validation loss results into near-optimal test-data performance for our benchmarks.
– *Number of RGCN Layers:* We evaluated how the number of layers (i.e., range `[1, 3]`) used for training the RGCN affects *OmniMatch*'s performance. Our results showed that using two layers provides the highest effectiveness gains, meaning that *OmniMatch*'s model benefits from looking one hop away from each node (column). This verifies our intuition that leveraging transitivity in the similarity graph improves the quality of the joinability predictions.
– *Number of Epochs:* We trained *OmniMatch* for several epochs and used loss curves with a `90:10` training/validation data split. Notably, using more than `30` epochs does not incur considerable changes in the training/validation losses. Therefore, for the rest of the experiments, we train *OmniMatch* for `30` epochs; the same stands for the Random Forest and MLP baselines.
– *Dimension of Embeddings:* We assessed the influence on *OmniMatch*'s effectiveness when producing column representations of varying dimensionality through the RGCN model. We ran experiments with {`32, 64, 128, 256, 512`} dimensions and found that column embeddings of 256 dimensions produce the best results.
– *Initial Node Features:* We evaluated how the initial node features we use for training the RGCN affect the performance of *OmniMatch*. Instead of using the proposed node features, we generated random feature vectors for each node of the same length as the RGCN's dimension of embeddings. Results verified the effectiveness of our node feature initialization process, as we observed a decrease of more than 10% in terms of PR-AUC scores when using randomized initial node features.
– *Loss Function:* As we discussed in Section 5, our training process can be guided using two different loss functions: *i*) cross-entropy loss and *ii*) triplet margin loss. Notably, the results show that using triplet margin loss can greatly improve the effectiveness as opposed to the cross entropy loss; its ability to bring closer column representations of joinable pairs while setting apart the ones of non-joinable pairs helps *OmniMatch* to better distinguish between the two cases.

Implementation details. For training, we use Adam [41] with a learning rate of `0.001`, while we use an MLP of one hidden layer when employing *OmniMatch* with a cross-entropy loss. *OmniMatch* is implemented in Python; for implementing the RGCN model we used the Deep Graph Library (DGL) [68] on top of PyTorch.

[4]https://www.sbert.net/



(a) City Government (b) Culture Recreation (c) Freyja (d) GDC

Figure 8: Effectiveness comparison of *OmniMatch* with other ML-models for various similarity thresholds.

Best F1 Scores			
Benchmark	***OmniMatch***	**Random Forest**	**MLP**
City Government	**0.857**	0.827	0.813
Culture Recreation	**0.894**	0.862	0.755
Freyja	**0.695**	0.671	0.290
GDC	**0.735**	0.621	0.069
PR-AUC Scores			
Benchmark	***OmniMatch***	**Random Forest**	**MLP**
City Government	**0.920**	0.805	0.788
Culture Recreation	**0.921**	0.835	0.618
Freyja	**0.589**	0.563	0.134
GDC	**0.679**	0.593	0.032

Table 3: Metrics comparison to other ML models.

6.2 Comparison to State-of-the-Art Baselines

In Figure 7, we show how *OmniMatch* compares against the state-of-the-art methods (Section 6.1) in terms of effectiveness using Precision-Recall curves. First, our method significantly outperforms the baselines since it can consistently provide high precision values even for recall values close to `0.8`, for the City Government and Culture Recreation benchmarks. Essentially, our method achieves high precision no matter the similarity threshold (except for very low ones), thus securing the quality of the returned joins. *OmniMatch*'s performance is consistently better than the rest of the methods also in the Freyja and GDC benchmarks; yet, we see a drop in precision since some of the fuzzy join cases in these benchmarks require domain knowledge. Interestingly, the column matching methods (COMA and DB) give low precision even for recall values that are not high, i.e., when the similarity thresholds are high. The reason is that these methods rely on a limited set of similarity signals based on data instances, which do not account for value semantics and syntactic differences, leading to false joinability predictions. Their results get worse in the Culture Recreation, Freyja and GDC benchmarks due to their more difficult join cases.

On the other hand, Starmie, with its contextualized column representations, does not deliver high precision for recall values above `0.6`. This mainly happens due to the counter-intuition behind contextualized column representations and join discovery: columns that join do not necessarily share similar contexts. In addition, the training examples produced by Starmie do not account for value discrepancies (i.e., fuzzy joins). Similarly, DeepJoin embeddings entail low precision for high recall, mainly due to the positive and negative pairs on which the model is trained, which are not guaranteed to be accurate. On the contrary, *OmniMatch* avoids this issue by relying on a training example generation that ensures true positive and negative pairs (Section 5.3). Furthermore, challenging join cases, where value overlaps are relatively small, are difficult to be captured by DeepJoin, since it cannot propagate similarity signals as our graph model. Both Starmie and DeepJoin show low performance when evaluated on the GDC benchmark, which we believe is mainly because their models cannot generalize in smaller repositories, with fewer training examples to leverage.

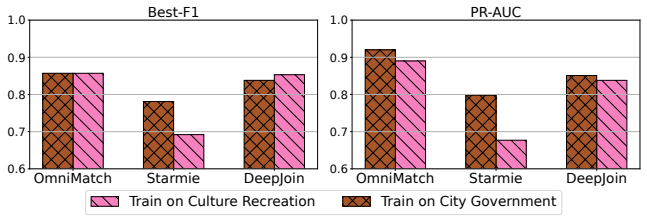
Best F1 and PR-AUC scores in Table 2 verify that *OmniMatch* is the most effective method across all benchmarks and similarity thresholds. It is also the most consistent one, as the performance of other methods fluctuates depending on the underlying datasets.

> ***Takeaways:*** *i)* OmniMatch *is consistently more effective than the state-of-the-art baselines, and ii) other methods exhibit low precision for high recall, while* OmniMatch *provides better predictions.*

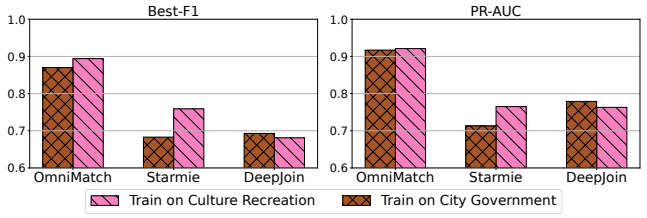
6.3 Comparison to Other ML Models

Figure 8 shows the Precision-Recall of our method compared to the Random-Forest (RF) and MLP models for all join benchmarks. We observe that *OmniMatch*'s joinability prediction model is superior to the other two, as it achieves considerably higher precision for higher recall values; the RF model achieves higher precision for lower recall values in both Freyja and GDC datasets, since it manages to capture simple join cases, while the MLP model considerably underperforms due to the low volume of training data. This result highlights the effectiveness of our graph modeling: *OmniMatch*'s RGCN column representations better capture column join relationships and avoid false positive predictions of models that rely only on the column pairwise similarities.

Results in Table 3 verify *OmniMatch*'s improvements in overall effectiveness as opposed to using less sophisticated ML models. Specifically, our method produces the highest overall F1-score, i.e., it can predict more accurate join relationships than pairwise similarities in conjunction with either an RF or MLP model. In addition, the high PR-AUC scores further showcase that *OmniMatch* consistently achieves high precision regardless of the similarity threshold used to decide whether a column pair represents a valid join. In contrast, using only the column pairwise similarities cannot help the RF and MLP models to capture less direct join relationships, while it can critically increase false-positive rates.



(a) Testing on City Government



(b) Testing on Culture Recreation

Figure 9: Effects on Best F1 and PR-AUC scores when the models of OmniMatch and other self-supervised methods are trained on different datasets than the ones tested on.

Takeaway: OmniMatch*'s prediction RGCN-based model leverages column pairwise similarities to result in significantly better effectiveness than less sophisticated prediction models.*

6.4 Model Effectiveness on Unseen Data

In Figure 9 we show the best F1 and PR-AUC scores achieved by *OmniMatch*, Starmie and DeepJoin when their models are trained on datasets different than the testing ones, to evaluate their learning capacity and robustness. We also show their scores when the models are trained on the same datasets they are tested on, to observe whether and how they are affected. To begin with, we see that *OmniMatch* provides with predictions that are the most accurate across all methods and test datasets. Moreover, its best F1 and PR-AUC scores are very slightly, if not, affected when trained on different datasets that the test ones. Essentially, *OmniMatch* can considerably preserve its high effectiveness, since during inference the model considers not only the profiles of a column pair but also the similarity signals between them. Therefore, we notice the importance of leveraging information from column pairwise similarity metrics, towards building joinability prediction models that can be robust to deviations between training and test data.

On the contrary, we observe that Starmie considerably underperforms when its model is applied on unseen data, since the training samples it fabricates are tightly tailored to the datasets in its input; hence, they do not help the model to abstract knowledge and effectively apply it on unseen data. Surprisingly, DeepJoin's results when tested on unseen data seem to slightly improve. We believe this to happen due to the false positives and negatives included in the training examples that might be propagated to testing. Specifically, when testing on the same datasets that the model was trained, column pairs that were falsely regarded as joins or non-joins during training will be inaccurately classified. Therefore, we see that DeepJoin's debatable training example generation process can cause the model to be inconsistent in terms of effectiveness.

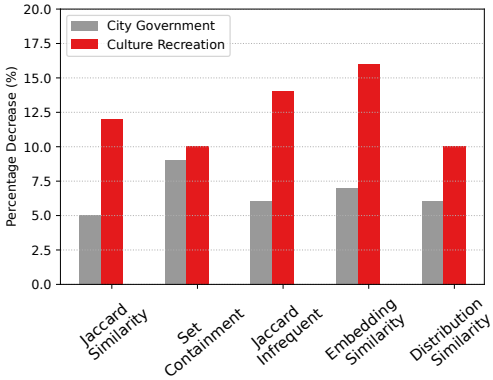


Figure 10: Reduction (in Percentage Decrease) of best F1-scores when *OmniMatch* considers a single similarity signal.

Takeaway: OmniMatch *can exhibit high effectiveness and consistent results even when tested on datasets unseen during training, due to the column profiles and similarity signals it employs.*

6.5 Ablation Study: Effect of Similarity Signals

We evaluate the power of using multiple similarity signals to construct our graph, in contrast to considering single ones. In Figure 10, we show the percentage decrease in the best F1-score achieved by *OmniMatch* when considering only one similarity signal per run. First, we see that the results support our intuition: using only one signal to build the similarity graph considerably affects the ability of our model to decide correctly on whether a column pair represents a join. Indeed, relying on single similarities incurs drops in the best F1 scores achieved due to increasing false positive rates. Moreover, many valid column join cases are not captured when employing single similarity signals due to information loss of transitive paths in the constructed similarity graph. For instance, using only Jaccard similarity can severely harm the effectiveness of capturing fuzzy joins since it checks only for exact value overlaps.

In addition, a crucial observation here is that the percentage decrease vastly relies on the underlying datasets and column joins to be captured. As we see in Figure 10, the drop in best F1-scores is significantly higher on the Culture Recreation benchmark with percentage decrease values of at least 10%. This is due to the following two reasons: *i*) there are column pairs in this benchmark that share (partial) value overlaps (e.g., dates), whereas they do not represent join relationships and *ii*) most column joins in the City Government benchmark are more distinguishable, i.e., a potential (partial) value overlap strongly indicates a valid join. Finally, no similarity signal consistently incurs larger/smaller effectiveness drops across the two join benchmarks. This observation reinforces our claim that no similarity signal can be fully trusted when isolated from the rest since its effectiveness depends on the characteristics of the underlying datasets. Only the complete set of similarity signals used in *OmniMatch* can provide the best join discovery results.

Takeaways: *i) using a single similarity signal incurs a notable decrease in effectiveness, and ii)* OmniMatch*'s consistency strongly depends on using all proposed similarity signals.*

Benchmark	T1 + T2	T3	I1	I2	Total
City Government	48.8	7	53.6	0.5	109.9
Culture Recreation	23	8.4	8.7	0.5	40.6

Table 4: *OmniMatch* execution times in minutes (CPU). T1-T3 and I1,I2 refer to different steps as shown in Figure 6.

6.6 OmniMatch Execution Times

While we consider joinability discovery as an offline procedure in data products, for the sake of completeness we report in Table 4 the execution times of *OmniMatch* for both join benchmarks. Specifically, we note down the time for the steps we show in Figure 6, *i*) generating joinable pairs and transforming them into a similarity graph (T1 + T2), *ii*) training *OmniMatch*'s joinability prediction model (T3), *iii*) building the similarity graph based on the original datasets (I1), and *iv*) using the trained model for inference on it (I2). As expected, we see that the main bottleneck of our method is the similarity graph construction both for training and inference: computing the set of similarity signals and the initial node features for all column pair combinations for different datasets entails numerous column pairwise operations; yet, accelerating these computations is a trivial issue that is not in the scope of this work. On the other hand, we see that training times in both benchmarks are relatively small, especially when we consider that training takes place on a CPU; notably, training of state-of-the-art column embedding methods [19, 23] requires access to a GPU. Finally, the discrepancies we observe between the two benchmarks are due to the different number of columns and values stored in them.

7 RELATED WORK

In this section, we discuss related work relevant to joinability discovery, including schema matching, dataset search/discovery and other graph-based solutions for data integration.

7.1 Schema Matching

Schema matching on tabular data includes automated methods for capturing relevance between columns of dataset pairs [58]. Such approaches might use metadata at the schema level [50, 53], rely on column instances to compute pairwise similarities [22, 75], or even leverage query logs [21]; in addition, there exist methods using ensembles of similarity measures [17, 62]. While effective on clean and homogeneous data, these methods struggle when there is discrepancy in the formats of values that two related columns store [45]. Multiple matching methods have emerged that leverage *embeddings*. To this end, some methods have directly employed pre-trained models to embed column names [26], schema metadata [77], and/or cell-values [20, 54], whereas others add pre-processing steps to adjust to input datasets [8, 25, 44]. However, both approaches seem to be insufficiently effective when used for matching related columns [45]. Magneto [49] leverages LLMs for pre-training smaller language models and re-ranking towards pairwise matching.

7.2 Related Dataset Search/Discovery

Related-dataset search methods focus on finding relevant tables with respect to a given/query table. Typically, such methods use column similarity signals (as used in schema matching methods [45]) between column pairs to generalize relatedness scores between datasets. Several works in this area attempt to capture *unionable* tables, i.e., tables that considerably overlap schema-wise to the one in the input. Table unionability works include methods that employ ensembles of various column-pairwise similarity measures among tables [6, 54], knowledge-bases [38] and pre-trained language models to capture column semantics [13, 23, 33]. More relevant to our work are the dataset search methods focusing on *joinability*, i.e., finding tables that can be joined with a given one. Earlier methods rely on simple metrics [15, 24, 27, 80], such as *Jaccard* similarity, to compute column-pairwise relevance; using similar measures, Freyja [52] attempts to quantify the quality of discovered joins. Recent dataset search methods towards joinability, rely on pre-trained language models [12, 18], or even try to build their own join prediction models based on them [19].

Contrary to top-k dataset search that focuses on returning the top-k datasets, given a query dataset, *OmniMatch* returns pairs of joinable columns. Yet, we draw inspiration from pairwise column similarities that have been used in related literature (Section 4.1).

7.3 Graph-Based Data Integration

Several graph-based solutions for different data integration problems have been introduced [3, 8, 24, 38, 44], with works employing GNNs showing promising results in tasks similar to schema matching and joinability discovery [30–32, 65, 69, 70]. GCNAlign [69] leverages GCNs for cross-lingual *knowledge graph alignment*. Entity alignments are discovered based on the distances between entities in the embedding space. NMN [70] introduces an *entity alignment* framework for tackling the structural heterogeneity challenge by estimating the similarities between entities to capture both the topological structure and the neighborhood difference. MEDTO [31] utilizes a hybrid GNN (i.e., RGCN and hyperbolic GNN) for data to *ontology matching*. KGLac [32] leverages GNNs to enable automatic graph learning for advanced and semantic *data discovery*. The authors in [65] apply GNNs towards medical entity disambiguation. On the other hand, FlexER [30] employs GNNs towards multi-intent entity resolution. Similar to the above approaches, *OmniMatch* models column relationships as a graph. To the best of our knowledge, *OmniMatch* is the first approach to model the problem of joinability discovery employing GNNs, and offering an improved solution to the problem compared to the current state of the art.

8 CONCLUSION

In this paper, we introduced *OmniMatch*, a novel self-supervised method that captures joinability relationships among tabular data in a data product. *OmniMatch* leverages a comprehensive set of similarity signals and the transitive power of a graph model to learn column representations based on an RGCN. Notably, our method can automatically generate positive and negative join examples to guide the learning process. Our experimental evaluation shows that *OmniMatch* is considerably more effective than state-of-the-art column matching and representation methods. In addition, *OmniMatch*'s model is substantially more accurate and generalizable than other joinability prediciton models. We also justify the gains of using the comprehensive set of similarity signals we propose.

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *The VLDB Journal* 24 (2015), 557–581.

[2] Foto N Afrati, Anish Das Sarma, David Menestrina, Aditya Parameswaran, and Jeffrey D Ullman. 2012. Fuzzy joins using mapreduce. In *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 498–509.

[3] Naser Ahmadi, Hansjörg Sand, and Paolo Papotti. 2022. Unsupervised matching of data and text. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1058–1070.

[4] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics* 11 (2023), 227–249.

[5] Sagar Bharadwaj, Praveen Gupta, Ranjita Bhagwan, and Saikat Guha. 2021. Discovering related data at scale. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1392–1400.

[6] Alex Bogatu, Alvaro AA Fernandes, Norman W Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *IEEE ICDE*.

[7] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013).

[8] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1335–1349.

[9] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.

[10] Zhimin Chen, Yue Wang, Vivek Narasayya, and Surajit Chaudhuri. 2019. Customizable and scalable fuzzy join for big data. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2106–2117.

[11] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2021. ARDA: Automatic Relational Data Augmentation for Machine Learning. *Proceedings of the VLDB Endowment* 13, 9 (2021).

[12] Tianji Cong, James Gale, Jason Frantz, H. V. Jagadish, and Çagatay Demiralp. 2023. WarpGate: A Semantic Join Discovery System for Cloud Data Warehouses. In *CIDR*.

[13] Tianji Cong, Fatemeh Nargesian, and HV Jagadish. 2023. Pylon: Semantic Table Union Search in Data Lakes. *arXiv preprint arXiv:2301.04901* (2023).

[14] Zhamak Dehghani. 2022. *Data mesh: Delivering Data-Driven Value at Scale*. O' Reilly Media.

[15] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. 2017. SilkMoth: an efficient method for finding related sets with maximum matching constraints. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1082–1093.

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

[17] Hong-Hai Do and Erhard Rahm. 2002. COMA—a system for flexible combination of schema matching approaches. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 610–621.

[18] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 456–467.

[19] 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 (jun 2023), 2458–2470. https://doi.org/10.14778/3603581.3603587

[20] Xingyu Du, Gongsheng Yuan, Sai Wu, Gang Chen, and Peng Lu. 2024. In Situ Neural Relational Schema Matcher. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 138–150.

[21] Hazem Elmeleegy, Mourad Ouzzani, and Ahmed Elmagarmid. 2008. Usage-based schema matching. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 20–29.

[22] Daniel Engmann and Sabine Massmann. 2007. Instance Matching with COMA++.. In *BTW workshops*, Vol. 7. 28–37.

[23] 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. *Proceedings of the VLDB Endowment* 16, 7 (2023), 1726–1739.

[24] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1001–1012.

[25] Raul Castro Fernandez and Samuel Madden. 2019. Termite: a system for tunneling through heterogeneous data. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–8.

[26] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 989–1000.

[27] Raul Castro Fernandez, Jisoo Min, Demitri Nava, and Samuel Madden. 2019. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1190–1201.

[28] Javier de Jesús Flores Herrera, Sergi Nadal Francesch, and Óscar Romero Moral. 2021. Towards scalable data discovery. In *Advances in Database Technology: EDBT 2021, 24th International Conference on Extending Database Technology: Nicosia, Cyprus, March 23-26, 2021: proceedings*. OpenProceedings, 433–438.

[29] Quan Gan, Minjie Wang, David Wipf, and Christos Faloutsos. 2024. Graph Machine Learning Meets Multi-Table Relational Data. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6502–6512.

[30] Bar Genossar, Roee Shraga, and Avigdor Gal. 2023. Flexer: Flexible entity resolution for multiple intents. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[31] Junheng Hao, Chuan Lei, Vasilis Efthymiou, Abdul Quamar, Fatma Özcan, Yizhou Sun, and Wei Wang. 2021. MEDTO: Medical Data to Ontology Matching Using Hybrid Graph Neural Networks. In *ACM SIGKDD*. 2946–2954.

[32] Ahmed Helal, Mossad Helali, Khaled Ammar, and Essam Mansour. 2021. A Demonstration of KGLac: A Data Discovery and Enrichment Platform for Data Science. *Proc. VLDB Endow.* 14, 12 (2021), 2675–2678.

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

[34] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1500–1508.

[35] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 3446–3456.

[36] Andra Ionescu, Kiril Vasilev, Florena Buse, Rihan Hai, and Asterios Katsifodimos. 2024. AutoFeat: Transitive Feature Discovery over Join Paths. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 1861–1873.

[37] Armand Joulin, Édouard Grave, Piotr Bojanowski, and Tomáš Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. 427–431.

[38] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. 2023. Santos: Relationship-based semantic table union search. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.

[39] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J Miller. 2022. Integrating Data Lake Tables. *Proceedings of the VLDB Endowment* 16, 4 (2022), 932–945.

[40] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.

[41] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[42] Sachin Konan, Larry Rudolph, and Scott Affens. 2024. Automating the Generation of a Functional Semantic Types Ontology with Foundational Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*. 248–265.

[43] Kezhi Kong, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Chuan Lei, Christos Faloutsos, Huzefa Rangwala, and George Karypis. 2024. OpenTab: Advancing Large Language Models as Open-domain Table Reasoners. In *The Twelfth International Conference on Learning Representations*.

[44] Christos Koutras, Marios Fragkoulis, Asterios Katsifodimos, and Christoph Lofi. 2020. REMA: Graph Embeddings-based Relational Schema Matching.

[45] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating matching techniques for dataset discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 468–479.

[46] Aristotelis Leventidis, Laura Di Rocco, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. 2021. DomainNet: Homograph Detection for Data Lake Disambiguation. *EDBT 2021* (2021).

[47] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples. In *Proceedings of the 2021 International Conference on Management of Data*. 1064–1076.

[48] 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* (2019).

[49] Yurong Liu, Eduardo Pena, Aecio Santos, Eden Wu, and Juliana Freire. 2024. Magneto: Combining Small and Large Language Models for Schema Matching. *arXiv preprint arXiv:2412.08194* (2024).

[50] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. 2001. Generic schema matching with cupid. In *VLDB*, Vol. 1. Citeseer, 49–58.

[51] Christopher Manning and Hinrich Schutze. 1999. *Foundations of statistical natural language processing*. MIT press.

[52] Marc Maynou, Sergi Nadal, Raquel Panadero, Javier Flores, Oscar Romero, and Anna Queralt. 2024. FREYJA: Efficient Join Discovery in Data Lakes. *arXiv preprint arXiv:2412.06637* (2024).

[53] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. 2002. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th international conference on data engineering*. IEEE, 117–128.

[54] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. In *VLDB*.

[55] Hoa Nguyen, Ariel Fuxman, Stelios Paparizos, Juliana Freire, and Rakesh Agrawal. 2011. Synthesizing Products for Online Catalogs. *Proceedings of the VLDB Endowment* 4, 7 (2011), 409–418.

[56] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[57] 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)*. 1532–1543.

[58] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal* 10, 4 (2001), 334–350.

[59] 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*. Association for Computational Linguistics. https://arxiv.org/abs/1908.10084

[60] Aecio Santos, Eden Wu, Roque Lopez, Sarah Keegan, Eduardo Pena, Wenke Liu, Yurong Liu, David Fenyo, and Juliana Freire. 2025. GDC-SM: The GDC Schema Matching Benchmark (1.0) [Dataset]. *Zenodo* (2025). https://doi.org/10.5281/zenodo.14963588

[61] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*. Springer, 593–607.

[62] Roee Shraga, Avigdor Gal, and Haggai Roitman. 2020. Adnev: Cross-domain schema matching using deep similarity matrix adjustment and evaluation. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1401–1415.

[63] Roee Shraga, Haggai Roitman, Guy Feigenblat, and Mustafa Cannim. 2020. Web table retrieval using multimodal deep learning. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. 1399–1408.

[64] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755* (2024).

[65] Alina Vretinaris, Chuan Lei, Vasilis Efthymiou, Xiao Qin, and Fatma Özcan. 2021. Medical entity disambiguation using graph neural networks. In *Proceedings of the 2021 international conference on management of data*. 2310–2318.

[66] Jiannan Wang, Guoliang Li, and Jianhua Fe. 2011. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 458–469.

[67] Jin Wang, Chunbin Lin, and Carlo Zaniolo. 2019. Mf-join: Efficient fuzzy string similarity join with multi-level filtering. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 386–397.

[68] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[69] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. 2018. Cross-lingual Knowledge Graph Alignment via Graph Convolutional Networks. In *EMNLP*. 349–357.

[70] Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, and Dongyan Zhao. 2020. Neighborhood Matching Network for Entity Alignment. In *ACL*. 6477–6487.

[71] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 97–108.

[72] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 8413–8426.

[73] Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Shen Wang, Huzefa Rangwala, and George Karypis. 2023. NameGuess: Column Name Expansion for Tabular Data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 13276–13290.

[74] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2010. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 805–814.

[75] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2011. Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 109–120.

[76] Shuo Zhang and Krisztian Balog. 2018. Ad hoc table retrieval using semantic similarity. In *Proceedings of the 2018 world wide web conference*. 1553–1562.

[77] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M Patel. 2023. Schema matching using pre-trained language models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1558–1571.

[78] Yi Zhang and Zachary G Ives. 2020. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1951–1966.

[79] Zixuan Zhao and Raul Castro Fernandez. 2022. Leva: Boosting Machine Learning Performance with Relational Embedding Data Augmentation. (2022).

[80] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*. 847–864.