Table Overlap Estimation through Graph Embeddings

FRANCESCO PUGNALONI, Hasso Plattner Institute, University of Potsdam, Germany
LUCA ZECCHINI, University of Modena and Reggio Emilia, Italy
MATTEO PAGANELLI, University of Modena and Reggio Emilia, Italy
MATTEO LISSANDRINI, University of Verona, Italy
FELIX NAUMANN, Hasso Plattner Institute, University of Potsdam, Germany
GIOVANNI SIMONINI, University of Modena and Reggio Emilia, Italy

Discovering duplicate or high-overlapping tables in table collections is a crucial task for eliminating redundant information, detecting inconsistencies in the evolution of a table across its multiple versions produced over time, and identifying related tables. Candidate duplicate or related tables to support this task can be identified via the estimation of the largest table overlap. Unfortunately, current solutions for finding it present serious scalability issues for heavy workloads: Sloth, the state-of-the-art framework for its estimation, requires more than three days of machine time for computing 100k table overlaps.

In this paper, we introduce Armadillo, an approach based on graph neural networks that learns table embeddings whose cosine similarity approximates the *overlap ratio* between tables, i.e., the ratio between the area of their largest table overlap and the area of the smaller table in the pair. We also introduce two new annotated datasets based on GitTables and a Wikipedia table corpus containing 1.32 million table pairs overall labeled with their overlap. Evaluating the performance of Armadillo on these datasets, we observed that it is able to calculate overlaps between pairs of tables several times faster than the state-of-the-art method while maintaining a good quality in approximating the exact result.

CCS Concepts: • **Information systems** → *Information retrieval*; *Information integration*; *Deduplication*.

Additional Key Words and Phrases: table representation learning, graph neural networks, data lakes, web tables, table overlap, table matching

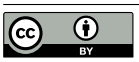
ACM Reference Format:
Francesco Pugnaloni, Luca Zecchini, Matteo Paganelli, Matteo Lissandrini, Felix Naumann, and Giovanni Simonini. 2025. Table Overlap Estimation through Graph Embeddings. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 228 (June 2025), 25 pages. https://doi.org/10.1145/3725365

1 Estimating table overlap

Tables are one of the most common formats used in data lakes and Web sources for organizing information that refers to a common schema [39]. For instance, a large amount of tabular data can be found in GitHub, which contains more than 10 million tables stored as CSV files [25], and Wikipedia, where more than 3 million HTML tables have been created throughout its history [2, 5].

In these large table corpora, it is frequent to find pairs of tables that share the same content across multiple cells, i.e., *overlapping* tables. This overlap, depending on its size and shape, can be a signal of table relatedness. For instance, a *vertical* overlap spanning several rows across a

Authors' Contact Information: Francesco Pugnaloni, Hasso Plattner Institute, University of Potsdam, Germany, francesco.pugnaloni@hpi.de; Luca Zecchini, University of Modena and Reggio Emilia, Italy, luca.zecchini@unimore.it; Matteo Paganelli, University of Modena and Reggio Emilia, Italy, matteo.paganelli@unimore.it; Matteo Lissandrini, University of Verona, Italy, matteo.lissandrini@univr.it; Felix Naumann, Hasso Plattner Institute, University of Potsdam, Germany, felix.naumann@hpi.de; Giovanni Simonini, University of Modena and Reggio Emilia, Italy, giovanni.simonini@unimore.it.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 2836-6573/2025/6-ART228
https://doi.org/10.1145/3725365

R	Monument	City	Country	Year
r_1	TV Tower	Berlin	Germany	1965-1969
r_2	Big Ben	London	England	1843-1869
r_3	St. Davids	-	England	1915-1921
r_4	Colosseum	Rome	Italy	70-80
r_5	AquaClaudia	Rome	Unknown	52
r_6	Well of Life	Zagreb	Croatia	Unknown

S	Nation	Population	City
s_1	England	8,866,180	London
s_2	Italy	2 752 908	Rome
s_3	England	1 348	St.Davids
s_4	Unknown	52	Hum
s_5	Croatia	Unknown	Rastoke
s_6	Germany	3.755.251	Berlin

Fig. 1. Pair of tables that overlap in different ways. ($\langle$"*Unknown*", "52"$\rangle$, $\langle$"*Croatia*", "*Unknown*"$\rangle$) has area four and is the largest overlap obtainable without reordering rows and columns, while ($\langle$"*Berlin*", "*Germany*"$\rangle$, $\langle$"*Rome*", "*Italy*"$\rangle$, $\langle$"*London*", "*England*"$\rangle$) is the real largest overlap, it has area six and is obtained through a permutation of rows and columns. The overlap ratio is 6/18 (largest overlap area / area of the smaller table).

few columns may denote a pair of joinable tables. Similarly, a *horizontal* overlap can be a sign of unionability. Further, an overlap covering many rows and columns might represent a case of duplicate tables, table containment, or different versions of the same table [3].

Detecting overlaps is a challenging task for traditional set-based methods for data discovery, as it requires considering the alignment of cell values in a table and the structure of the tables themselves. To fill this gap, the concept of *largest overlap*, i.e., the largest common *rectangular* subtable existing between two tables under any row and column reordering, was recently introduced [54]. We present an illustrative example of largest overlap in Figure 1.

Identifying tables for which the largest overlap exceeds a certain area and/or presents a specific shape finds application in multiple real-world scenarios, from the detection of largely overlapping tables in a data lake, e.g., for subsequent operations of change propagation [3] or elimination of redundant information, to more specific tasks, e.g., the detection of copying dependencies across multiple sources [35] or the automated discovery of multi-column joins [54]. In addition, table overlap not only serves user-facing applications, but can also be used as a filtering or blocking method for downstream tasks, such as table reclamation [16].

In practice, all proposed use cases fall under two main scenarios: *(i) table matching*, i.e., retrieving all the clusters of related tables in a corpus, and *(ii) table querying*, i.e., given a *query table*, identifying a set of related tables in a corpus by building a ranking or using a similarity threshold. Since the first scenario requires comparing all possible pairs, it has a quadratic complexity, and even when working with datasets of a few thousand elements, a large number of comparisons needs to be performed. The second, instead, is linear with respect to the number of tables in the corpus. However, it can be equally challenging if computed in a brute-force manner, indeed Sloth [54], the only prior approach available for estimating the largest table overlap, takes more than two hours of machine time to compare 1 table with a data lake of 10K tables. Thus, in this paper, we analyze the problem of efficiently approximating the *maximum normalized table overlap*, referring to it as *overlap ratio*.

1.1 Table overlap ratio

We define the overlap ratio between two tables by extending the definition of the largest table overlap [54]. The intuition is to identify the sub-tables of maximum area, in terms of number of cells, that are shared between the two tables and normalize their area with respect to the area of the smaller table.

Given two tables R and S defined on the schemas X and Y respectively, we first define an attribute mapping M as the bijective function that links a subset of columns of the first table $X_M \subseteq X$ to another set $Y_M \subseteq Y$ of the second table. Each mapping determines a table overlap O_M, which corresponds

to the sub-table obtained from the intersection under bag semantics of the tuples in the columns linked by the mapping. We call the overlap area for the mapping M the table area of O_M, i.e., the number of cells contained in the table overlap generated by the mapping M. Among all the possible mappings, we are interested in identifying the ones with the maximum overlap area.

Example 1 (Largest table overlap). *Given the tables R and S in Figure 1 it is possible to identify several attribute mappings. Just to list a few, the mapping* M_1:⟨*Year*⟩→⟨*Population*⟩ *generates an overlap area of two, i.e., only two cells are shared between the two columns. The mapping* M_2:⟨*City, Country*⟩→⟨*City, Nation*⟩ *instead, represents a mapping of area six and since there are no other mappings that generate a larger area, it is a largest table overlap.*

Formally, we call $O(R, S)$ the set of table overlaps determined by all possible attribute mappings between R and S. We instead identify with $O^*(R, S) \subseteq O(R, S)$ the set of table overlaps with the largest overlap area, and their respective areas with $A^*(R, S)$.

Definition 1.1 (Overlap ratio). Let R, Y be tables defined on the schemas X, Y, A_R and A_S their table areas, and $A^*(R, S)$ their largest overlap area. We call

$$\theta(R, S) = \frac{A^*(R, S)}{min(A_R, A_S)} \in [0, 1]$$

overlap ratio between the tables R and Y.

Example 2 (Overlap ratio). *Table R and S have an area of* 6×4 *and* 6×3 *respectively, and their largest table overlap, deriving from* M_2*, is 6. Their overlap ratio is* $\theta(R, S) = \frac{6}{min(6\times4, 6\times3)} = \frac{6}{18} = 0.33$.

1.2 Efficient overlap ratio approximation

Although Sloth [54] provides a method to compute the largest overlap between pairs of tables, it has serious scalability issues. A possible approach to solving this problem is to consider methods that further approximate this calculation, such as considering tables as flattened sets of values and adapting a set-wise similarity measure for this purpose. The *Jaccard Similarity* [27] is generally used to compare textual data and defines the similarity of sets of elements as the ratio between the cardinality of their intersection and their union. It can be used to approximate the overlap ratio by finding the intersection under set semantics of values in a pair of tables and dividing their cardinality by the cardinality of their union. Since this approach uses the set semantics, it does not take into account the repetition of values in the tables. A solution to the problem is computing the Jaccard similarity under bag semantics, i.e., considering tables as bags of values that allow repetitions and dividing the cardinality of the bag intersection by the cardinality of the bag union. Since the number of values in the bag intersection is at most equal to the sum of the sizes of the two bags, it can predict at most a value of 0.5, hence, it cannot be used as it is. Two solutions are (1) scaling its result to the right interval by multiplying its prediction by two or (2) dividing the cardinality of the bag intersection by the size of the smaller bag.

Nonetheless, none of these approaches consider the structure of the tables and thus they fail when the table pairs have more than one common sub-table. In particular, the last method assumes that all values in the bag intersection are part of the same largest common subtable, always predicting an overlap greater or equal to the real one.

Example 3 (Approximate overlap ratio). *For the tables in Figure 1, the Jaccard similarity would predict an overlap of* $\frac{10}{27} = 0.37$ *under set semantics, an overlap of* $2 \cdot \frac{12}{42} = 0.57$ *under scaled bag semantics, and* $\frac{12}{18} = 0.67$ *under bag semantics normalized by the smaller table area. The true value of the overlap ratio between the tables is* $\frac{6}{18} = 0.33$*, and is overestimated by all methods.*

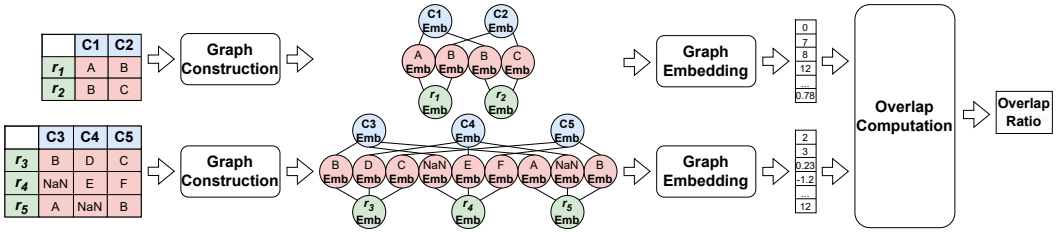


Fig. 2. Overlap Computation Pipeline

To address the challenge of effectively and efficiently estimating the largest table overlap between two tables, we introduce Armadillo, an approach that uses an intermediate graph representation and graph neural networks to generate table embeddings that encode not only information about the content of the tables but also about their structure—to capture the information about cell values appearing together in rows and columns. Thus, thanks to this vectorial representation, table representations can be compared efficiently to obtain an estimation of the overlap ratio between the original tables.

We also introduce two new datasets based on GitTables and a Wikipedia table corpus, containing 1.32 million table pairs overall labeled with their overlap, containing respectively 700k pairs of tables and 620k pairs of tables labeled with the value of their overlap ratio as computed by Sloth [54]. We further split this data into train/valid/test sets by (i) making sure to remove common tables between training and inference pairs, and (ii) balancing the distribution of the overlap ratio into ten equally spaced bins in the interval [0, 1]. In summary, our main contributions are:

- The Armadillo approach, which introduces the first GNN architecture for an efficient and effective approximation of the overlap ratio between tables[1]
- An extensive evaluation of our approach to the scenarios of table matching and table querying, highlighting its strengths and weaknesses
- Two open-source annotated datasets based on GitTables and a Wikipedia table corpus, comprising a total of 1.32 million table pairs with their true overlap ratio

2 Armadillo

Armadillo is a novel framework for the approximation of the overlap ratio between tables that exploits table embeddings generated through intermediate graph representations of tabular data. Given a pair of tables, the process of predicting their overlap consists of three major steps that are shown in Figure 2:

(1) *Graph construction*: the input tables are mapped to graphs, whose nodes and edges represent tabular elements and their structural relationships
(2) *Graph embedding*: the graphs are fed into a *graph neural network* (GNN) that computes graph embeddings, i.e., embeddings of the graph representation of the table contents
(3) *Overlap computation*: the graph embeddings are compared to predict the overlap ratio between the tables

In the following, we describe these steps, the implementation choices, and their motivations.

[1]Armadillos are cousins of Sloths, with a top speed of 48km/h, as opposed to 1km/h.

2.1 Graph construction

To determine the overlap ratio of a pair of tables, three key pieces of information are needed: which cells contain the same values, the areas of the tables, and the structure of the tables, i.e., which columns and rows the cells belong to. This information should be considered regardless of the order of the rows and columns in the input tables.

For intermediate representations of tables, two main approaches have been explored in the literature: (i) serializing tables into plain text or (ii) building graph-based representations. While the first approach cannot exhaustively encode the table structure, previous works [8, 20, 34] suggest that representing tables as graphs when generating embeddings of schema elements improves the expressivity of the final result. The intuition of these approaches is representing the schema elements and their relationships as nodes and edges, e.g., nodes can refer to cells, columns, and rows, while edges model relationships between them. Starting from this general scheme, several variants can be considered. Li et al. [34] map cells into one or more nodes in the graph depending on the number of tokens contained in the value, e.g., the number of words for sentences. Cappuzzo et al. [8] collapse different occurrences of the same value in a table into a single node. Ge et al. [20] do not use a specific node for the columns and insert all column-related information in the edges that link row and cell nodes.

None of these structures can be directly applied to our task. Dividing multi-token cells into multiple nodes makes it harder to identify exact matching cells. Compressing multiple repetitions of the same value across multiple cells into a single node limits the information on the multiplicity of values and the area of the table, since the overlap ratio is computed as a ratio of two areas this information plays an important role in estimating it. Representations that insert all column-related information in the edges that connect row and value nodes proved to perform well on entity matching tasks, but they can produce a not fully connected graph, limiting the expressive capabilities of embedding methods that exploit paths between nodes to propagate signals, e.g., graph neural networks.

Armadillo builds for each table a graph with the structure shown in Figure 2. It has three categories of nodes, one for each tabular element: column nodes, row nodes, and cell nodes. There is one distinct node for every row, column, and cell in the table. The graph is tripartite and undirected, hence, two categories of edges link cell nodes with the nodes associated with the rows and columns to which they belong. The topology of the graph depends solely on the number of rows and columns in the table. Hence, it provides information about the area of the table and its structure, but it does not take the actual values inside the cells into account. To provide information about the content of the table, each node is associated with an embedding. In this way, Armadillo generates a graph containing all the information to calculate the overlap ratio listed at the beginning of Section 2.1.

Node embeddings. Node embeddings need to represent different properties depending on the category of nodes. Embeddings of cell nodes must provide information about their values, encoding whether two cells have the same content or not. Word embedding approaches, e.g., word2vec [38], despite being time-efficient, are vulnerable to out-of-vocabulary tokens that cause the loss of information about the content of the cells. N-grams-based approaches, e.g., Fasttext [7], manage to reduce the out-of-vocabulary occurrences by generating embeddings of sub-words, but they cannot handle numerical data well enough. Transformer-based approaches, e.g., BERT [11], exploit more advanced tokenization techniques, e.g., WordPiece [49], solving the out-of-vocabulary problems also when working with numerical data, but they are computationally more demanding and slower, see Section 4.2.

Armadillo generates the initial embeddings of the cell nodes by hashing their values with the well-known SHA-256 algorithm. Since this hashing approach expects textual data as input, all the

numerical values are converted to strings. As the NaN values are considered when determining the largest overlap, they are converted to the string *null* and hashed like the other values. Since the output of the SHA-256 algorithm is a hexadecimal string composed of 64 characters, i.e., a sequence of 32 numbers that range from 0 to 255, the initial node embeddings have 32 dimensions.

This hashing approach is faster than transformer-based methods and, in the absence of collisions, generates the same representation for two cell nodes only if they contain the same value. As for the purpose of computing the overlap ratio the semantics of the values inside the cells is irrelevant, their hashes contain all the necessary information.

The initial embeddings of row and column nodes are generated by aggregating the embeddings of the cells that they contain. Minimum and maximum functions are not suitable aggregation functions for this purpose, as they do not provide information about all the cells within the row or column. The sum function offers a global representation of rows and columns but does not ensure embeddings with values from 0 to 255, generating a representation inconsistent with the one used for the cell nodes. To aggregate the cell nodes, Armadillo uses the mean aggregation function, which provides global information and generates embeddings with values in the correct range.

Graph construction. Algorithm 1 summarizes the sequence of operations that Armadillo performs to convert a table into a graph. Graphs are represented by two matrices containing the node embeddings and the edges in the graph, respectively. The node embedding matrix associates each node with an index and an embedding, and the edges are stored as pairs of indexes-values.

Given a table as input, Armadillo iterates over its rows and columns (lines 5-17). Each cell is converted to text and hashed by the *hash_string* function to obtain the cell embedding (lines 7-12). The cell node is then inserted into the graph and connected with the nodes related to the row and column it belongs to (lines 13-15). Furthermore, a reference to the cell embedding is stored in two intermediate variables, namely rows_to_values and columns_to_values, to subsequently have access to all the embeddings related to a given row or column (lines 16-17). Finally, Armadillo generates the embeddings of the columns and rows respectively by calculating the average of the embeddings of the related cells (lines 18-19).

As Armadillo must process all cells in the table, one by one, to construct the graph, the computational complexity grows linearly with the area of the table. If r is the number of rows and c the number of columns, the computational complexity of the algorithm grows linearly with the number of cells in the table, or more accurately $O(r \cdot c)$.

2.2 Graph embedding

The graph described in Section 2.1 is only an intermediate representation of the table, and needs further processing to become the final one. Armadillo carries out this operation in two steps. First, it refines the initial node embeddings using the connections between nodes in the graph. Then, it aggregates the node embeddings into a single final embedding, i.e., a graph embedding, which contains all the necessary information for approximating the overlap ratio.

The intuition of the embedding refinement process is to modify the embedding of a node by regarding its neighborhood in the graph. Given how the graph is constructed, this operation allows the framework to encode knowledge about the structure and content of the tables. We considered two possible approaches for this operation. The first one adapts EmbDI, a framework proposed by Cappuzzo et al. [8], which uses node2vec [21], and the second one relies on graph neural networks. Our experiments showed that, when it comes to working with large collections of tables, EmbDI is too slow, for this reason, Armadillo uses the second method (see Section 3.3 and Section 4 for further details). For the aggregation method, we apply a readout layer based on mean pooling [50].

Algorithm 1: Graph Construction Algorithm

```
Input: A table
edges ← init_edges()
embeddings ← init_node_embeddings()
rows_to_values ← init_rows_values_mappings()
columns_to_values ← init_columns_values_mappings()
foreach r ∈ table.rows() do
    foreach c ∈ table.columns() do
        v ← table.get_cell_value(r, c)
        if is_number(v) then
            v ← str(v)
        if is_na(v) then
            v ← "null"
        v_emb ← hash_string(v)
        embeddings.add_embedding(v_emb)
        edges.add_edge(r, v)
        edges.add_edge(c, v)
        rows_to_values[r].add(v)
        columns_to_values[c].add(v)
embeddings.compute_column_embeddings(columns_to_values, strategy = Mean)
embeddings.compute_row_embeddings(rows_to_values, strategy = Mean)
Output: embeddings, edges
```

Why graph neural networks? Node embedding approaches based on node2vec do not process graphs natively and exploit random walks to generate training sentences for a skip-gram model, finding node embeddings that are based on the topology of the graphs without considering possible features associated with the nodes. Graph neural networks are specifically designed to process graphs and require as input for every node a feature vector that contains information about it. Furthermore, GNNs are faster than word2vec since they do not need to train a skip-gram model from scratch for inference, a requirement that node2vec has instead.

In our framework we use *GraphSAGE*, a GNN architecture proposed by Hamilton et al. [23]. During our preliminary experiments, we defined lists of candidate values for the model hyperparameters, e.g., the number of layers and the embedding size, and we trained several versions of *GraphSAGE*. In Armadillo we use the configuration that performed the best. Given as input an edge list and a feature matrix, i.e., the embedding matrix, *GraphSAGE* is able to fine-tune the initial embeddings using information taken from the nodes' neighborhood, the refined node embeddings have 300 dimensions and their values are not limited to the interval [0,255]. In any case, Armadillo is modular, so one may change its configuration at will and retrain it to use different GNN models or a different number of dimensions for the final embeddings.

In the next paragraph, we present the concept of *receptive field* and explain how it influenced the choice of the number of layers.

Receptive field. The receptive field of a node is defined as the set of nodes used to generate its embedding. The cardinality of this set is mostly influenced by the number of layers of the network, specifically, if there is one layer, the embedding of a node is computed by aggregating the embeddings of the nodes at distance 1, if there are two layers, also nodes at distance 2, and so on.

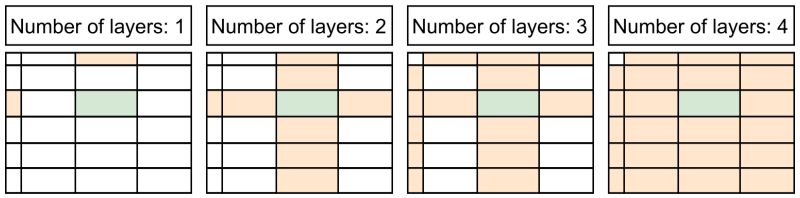


Fig. 3. Receptive field of a cell

A low number of layers would limit the amount of information that a node can receive from its neighbors. On the other hand, a high number of layers risks increasing the receptive field too much and cause *over-smoothing*: "the exponential convergence of suitable similarity measures on the node features" [41]. It is an effect that reduces the expressivity of the node embeddings.

Figure 3 visualizes the effect of an increasing number of layers on the receptive field by showing which schema elements influence the embedding of a cell node. When there is only one layer, the embedding of a cell node is computed by aggregating the embeddings of its row and its column, with two layers, the embeddings of the values in the same row or column are included, with three layers, also the *perimeter* of the table, i.e., all the row and column nodes, is included, and with four layers, the aggregation involves all the nodes in the graph.

When embedding row and column nodes, the situation is slightly different: with three layers, their receptive field covers the entire graph, but the overall pattern is similar. Our initial experiments suggested that using three layers provides the best performance for our use case.

2.3 Overlap computation and training

In this section, we explain (i) the similarity measure that we use to compare the embeddings and (ii) an overview of the training approach.

Cosine similarity. By Generating table embeddings, we map the problem of measuring a similarity between tables, to measure a similarity between vectors. Since the overlap ratio itself is a ratio between two areas, it is not suitable to compare vectors. A common choice for comparing embeddings is the *cosine similarity* [8, 55], but since it has values between -1 and 1 and the overlap ratio ranges from 0 to 1, it cannot be used as it is. For this reason, Armadillo uses a thresholded variant of the cosine similarity, where all the negative values of cosine similarity become 0, mapping the domain in the correct interval. Formally, let E_1, E_2 be table embeddings, $\theta_{predicted}(E_1, E_2) = \max(0, cos\_sim(E_1, E_2)) \in [0, 1]$ is their predicted table overlap ratio.

Training approach. Armadillo needs to learn the weights of the graph neural network to be functional. This process requires a dataset of triples containing pairs of tables and their overlap ratio. We provide two labeled datasets that are balanced by overlap intervals. We divide both into train, test, and validation without repetition of tables between the three sets. We further describe their structure and the approach used to build them in Section 3.2.

The training loop is composed of four steps that repeat for each training sample. The first step is the graph construction, where Armadillo maps pairs of input tables into graphs with the structure proposed in Section 2.1. For optimization reasons, Armadillo prebuilds graphs for all the tables in the dataset and in practice, this step only retrieves the correct pair of graphs from a dictionary. The second step is the embedding generation. Here, the model uses the GNN model followed by the aggregation layer to convert the graphs into table embeddings, as explained in Section 2.2. Keep in mind that even though during the training the embeddings are computed for pairs of tables, each table embedding is independent of the other, and each table has the same embedding, regardless of

Table 1. Stats for 256 834 GitTables and 128 620 WikiTables.

Dataset		# cols	# rows	area	# distinct	# missing
GitTables	mean	13.03	151.69	1351.85	520.80	118.56
	min	1.00	1.00	2.00	1.00	0.00
	max	698.00	19 254	99 856	99 558	98 908
WikiTables	mean	7.92	9.60	74.57	36.58	5.68
	min	1.00	1.00	1.00	1.00	0.00
	max	722	854	4350	1785	1573

which is the table it is compared to. The third step is the loss computation. We considered using two different losses in this step: the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). Our initial experiments suggested that MAE performed slightly better than MSE. Furthermore, its results are more interpretable. The fourth and last step is the backpropagation of the loss and the update of weights.

We trained two main models on two different domains of tables: *Armadillo-G*, trained on tables extracted from GitHub, and *Armadillo-W*, trained on web tables extracted from Wikipedia. We provide details on and links to both datasets in Section 3. We used 100 epochs of training for both of them, the first one needed almost 30 hours of training time, while the second one, due to the smaller tables, needed almost 7 hours in total.

3 Experimental setup

In this section, we describe the experimental setup by providing details on the datasets, baselines, and testing scenarios used in the evaluation. Since we are analyzing the task of estimating the overlap ratio between pairs of tables, we reorganized our benchmarks to obtain pairs of tables labeled with their overlap. This represents the ground truth for our evaluation. Then, we used these datasets to train two versions of Armadillo and to generate testing data for two evaluation scenarios (1) *table matching*, i.e., determining the overlap between pairs of tables, and (2) *table querying*, i.e., given a *query table*, identifying overlapping tables inside a table repository. We open-sourced both the datasets and the code used in this evaluation in a GitHub repository[2].

3.1 Datasets

GitTables. GitTables [25] contains 1 million tables extracted from GitHub. It was created to provide a benchmark containing tables that are structurally different from Web tables and resemble relational database tables. After removing empty and corrupted tables, we extracted a subset of approximately 256k tables from the original dataset. We report their statistics in Table 1. This dataset is diverse in composition, on average each table has 152 rows, 13 columns, and an average area of almost 1.4k cells, while some outlier tables can reach up to 19k rows and 698 columns with a maximum area of up to almost 100k cells. Further, repetitions of values inside the tables are relatively frequent: the average number of distinct values in a table is 520, which is almost half of the average area. Missing values, i.e., empty cells, are also frequent, and on average, there are 118 empty cells per table, i.e., approximately 10% of the area.

WikiTables. We denote as *WikiTables* a set of almost 2.13 million tables extracted from a dataset containing the different versions of the English Wikipedia tables collected throughout their history [4, 5]. In particular, we consider the tables from the latest snapshot available in the dataset,

[2]Link to the repository containing the artifacts produced.

dated September 1, 2019. After filtering out ill-formed and empty tables, we extracted a subset out of the dataset, for which we provide statistics in Table 1. Each table contains on average 10 rows, 8 columns, and has an area of 75 cells, while there are outlier tables reaching more than 800 rows, 700 columns, and an area of more than 4k. Further, the average area of the tables is less than 10% of the average area of tables in GitTables. Repetitions of values inside the tables are relatively frequent: the average number of distinct values in a table is 37, while the average table area is 75. Similarly to GitTables, the average number of missing values per table is less than 10% of the average table area. Nonetheless, there are tables containing up to more than 1k missing values.

3.2 Ground truth generation

We divided both GitTables and WikiTables into three disjoint sets of tables with a 3:1:1 ratio, containing respectively tables for the training, validation, and test sets. This was done to (1) prevent data leakages, i.e., to avoid inserting in the test and validation sets tables that are already included in the training set, and (2) evaluate the ability of the approach to generalize to unseen tables.

Secondly, for each set of tables, we identified pairs of tables with a uniform distribution of their overlap, i.e., we wanted to obtain an equal amount of pairs with high overlap ratio and low overlap ratio. To achieve this, we used a procedure that combines the LSH banding technique [33] with a random sampling method. The intuition is to use LSH to obtain pairs of tables with medium-high overlap and random sampling to select pairs of tables that share little information[3]. In particular, we performed LSH by varying the number of bands. We considered 8, 16, and 32 bands, determining estimated Jaccard similarity thresholds of about 0.88, 0.71, and 0.42, respectively. Then, we used Sloth [54] to compute the overlap for these pairs of tables. We then divided the pairs into ten equally spaced bins by overlap ratio, i.e., from 0 to 1 with a step of 0.1, and selected from each of them 50k pairs in the training set and up to 10k pairs in the validation and test sets. The outputs of this procedure were two collections of table pairs, balanced with respect to their overlap ratio. The *GitTables triple dataset* contains 500k training triples and 100k testing and validation triples, while the *WikiTables triple dataset* contains 500k training triples and 60k test and validation triples. For our evaluation, we trained and tested both within each dataset and also across the two datasets.

3.3 Baselines

We compare the performances of Armadillo with twelve other methods, eight of those based on table representation learning (TRL). For each method, we evaluate both the runtime and the overlap estimation effectiveness of the model.

The TRL baselines adapt existing sentence, graph, or tabular elements embedding methods to generate table embeddings whose cosine similarity approximates the overlap ratio between pairs of tables. We repurposed each model for generating one single table embedding instead of, e.g., one embedding for each cell. Since the cosine similarity between these table embeddings is not granted to approximate the overlap ratio between tables, for each model we trained a *scaling head*, i.e., a one-layer network that, given an embedding of a table, produces a new embedding with the desired properties. Each scaling head is trained on the same train, test, and validation data as Armadillo for a total of one hundred epochs.

Non TRL based. The first non-TRL baseline we compare against is **Sloth [54]**, the state-of-the-art method for identifying the largest overlap between tables. Sloth is an approximate method that has been proven to produce results that are exact or very close to exact. Thus, we also use Sloth's predictions as ground truth and compare to it only in terms of efficiency.

[3]The use of random sampling is justified by the fact that the distribution of the table pair similarity is more shifted towards low similarity values.

The second baseline is the **Jaccard similarity [27]**, a popular text similarity measure that can be repurposed for the overlap ratio estimation scenario. In this setting, a table is considered as a flattened set of values, i.e., a collection of the distinct values contained into its cells. More formally, let R, Y be two tables and $S(R), S(Y)$ the sets of the values contained into their cells. We call $\text{Jaccard}(R, Y) = \frac{|S(R) \cap S(Y)|}{|S(R) \cup S(Y)|}$ the Jaccard similarity between tables R and Y.

A *bag* is an unordered set of elements that allows repetitions, the Jaccard similarity under bag semantics is computed by dividing the cardinality of the intersection of two bags by the cardinality of their union and has values between 0 and 0.5. To keep track of the multiplicity of values the third approach (**Jaccard-B-U**) considers tables as flattened *bags* and scales Jaccard's result in the right interval by multiplying it by two. More formally, let R, Y be two tables and $B(R), B(Y)$ the bags of the values contained into their cells. We call $\text{Jaccard-B-U}(R, Y) = 2 \cdot \frac{|B(R) \uplus B(Y)|}{|B(R)| + |B(Y)|}$ the Jaccard similarity under bag semantics between tables R and Y.

The fourth one is another adaptation of the Jaccard similarity under bag semantics that normalizes the set intersection dividing it by the number of cells of the smaller table. We refer to it as **Jaccard-B-S**. More formally, let R, Y be two tables and $B(R), B(Y)$ the bags of the values contained into their cells. We call $\text{Jaccard-B-S}(R, Y) = \frac{|B(R) \uplus B(Y)|}{\min(|B(R)|, |B(Y)|)}$ the Jaccard similarity under bag semantics normalized by the smaller table area between the tables R and Y.

Sentence embedding based. State-of-the-art table representation learning baselines use adaptations of two sentence embedding models: BERT [11] and RoBERTa [36]. Both models, given an input tokenized sentence, generate context-aware embeddings of the tokens. We developed three pairs of embedding models that follow different approaches for generating table embeddings.

The first two models are **BERT-R** and **RoBERTa-R**, given an input table, they generate a sentence for each row by concatenating all their values and separating them with commas. Then, for each row, they generate token embeddings and average them to obtain a row embedding. Finally, they find the table embedding by averaging the row embeddings.

The models in the second pair are **BERT-T** and **RoBERTa-T**. Given an input table, they generate a sentence for each row in the same way as the previous two models. Then, they concatenate the row sentences using the newline character as separator to obtain a single sentence. Finally, they generate token embeddings for the tokens inside the table sentence and average them to obtain the table embedding.

The models in the third pair are **BERT-T-N** and **RoBERTa-T-N**. They follow the same embedding procedure as the models in the second pair but, before converting the table into a sentence, they perform a *noising operation* on all its values to measure the impact of the domain knowledge of the models on the embedding quality. The noising is performed by substituting the value of each cell with its hashed value obtained using the SHA-256 algorithm.

While it would be possible to generate one sentence for each cell value and then aggregate them into a final embedding, we excluded this baseline because, after initial investigation, we determined that (1) the embedding generation was extremely slow and (2) our experiments suggest that processing the entire table at once outperforms processing its components separately.

Cell and entity embedding based. TURL [10] is a table representation method that, given a parsed table, is able to provide embeddings of its components. We used the cell-filling version of TURL [10] to build a baseline that we call **TURL-T**. The model, given an input table, first, uses TURL to generate an embedding for each cell and distinct entity inside the table, then, averages these embeddings to obtain a table embedding.

Graph embedding based. EmbDI [8] exploits an intermediate graph representation and an adaptation of node2vec [21] to generate embeddings of rows, columns, and cells of pairs of tables. We

used it to develop our final baseline: **EmbDI-T**. The model, given a *pair* of tables, uses EmbDI to generate embeddings of their elements, and then, generates the table embeddings by averaging them. Differently from all the other embedding approaches, the embedding of a table changes if it is compared to two different tables, hence, while using this approach, there is no advantage in pre-computing the embeddings of the tables, limiting its usefulness in a real-world scenario.

3.4 Testing scenarios

We compare Armadillo with the competitors in two scenarios. The first concerns *table matching*, i.e., calculating the overlap ratio between each pair of tables in a given collection. The second is *table querying*, where the goal is identifying inside a repository tables that overlap with a given query table.

These scenarios test the effectiveness of the models in two different ways, by measuring their overall capabilities while approximating the overlap ratio and evaluating their ability to correctly identify the tables with the highest overlap.

For both scenarios, we also evaluate the efficiency of the approaches, by measuring the time needed to calculate the overlap ratio between a single pair of tables and the time needed to answer a query. We provide more details for each of these scenarios below.

Table matching. We consider the pairs of tables included in the test sets of the analyzed datasets. For each pair of tables, we calculate the overlap ratio with the considered approaches and measure the time necessary to carry out this operation. For Armadillo and its TRL competitors, we also apply a cross-domain evaluation by analyzing their performance when trained on WikiTables and tested on GitTables, and vice versa. This allows us to understand the generalization capabilities of the approaches to different domains.

Table querying. We simulate this scenario by considering 100 query tables and a collection of 10k tables. The choice of these dimensions, as discussed in Section 4.2.2, is motivated by the time needed for Sloth to execute: it required a total time of more than one week to complete the task on a single dataset. The query tables and the collection are extracted from the GitTables dataset by random sampling to simulate a real-world scenario, taking care to not select tables already seen by Armadillo during its training or validation.

We also included a different querying scenario: the table reclamation [16]. Given a query table, it aims to identify a set of generating tables inside a data lake to reconstruct the query table by performing both data discovery and integration. Since Gen-T [16], the state-of-the-art framework for table reclamation, does not scale well with large amounts of data, we propose Armadillo as a possible blocking method to reduce the number of candidate generating tables inside a data lake for a query table. In our experiments, we used a data lake composed of the tables inside the WikiTables dataset and the 32 data lake-tables inside the TP-TR small benchmark [16], the query table set is composed of the 26 query tables of TP-TR small.

4 Experiments

In the following, we first study the effectiveness of Armadillo in approximating the overlap ratio and then its efficiency.

Armadillo is implemented in Python 3.11.3 and its code is publicly available on GitHub. Tables are stored as CSV files divided into folders. Our experiments were performed on a server machine equipped with a AMD EPYC 7702P CPU with 64 cores @2-3.35GHz, two GPUs, one NVIDIA Quadro RTX A6000 with 48 GB of VRAM and one NVIDIA Quadro RTX 5000 with 16 GB of VRAM, and 512 GB of RAM, running Ubuntu 20.04.

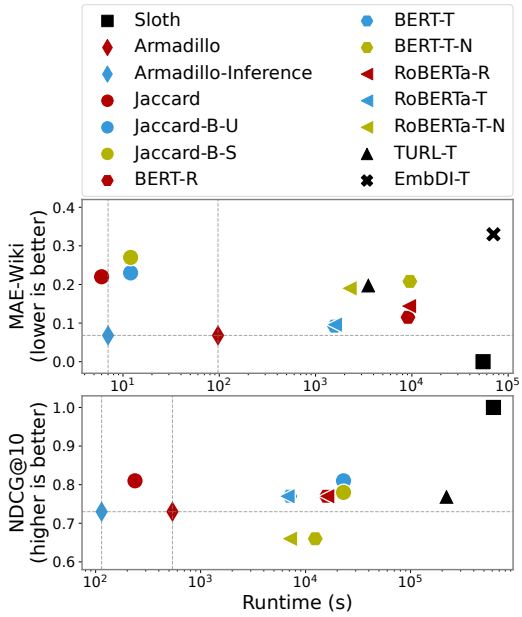


Fig. 4. Tradeoff between effectiveness and efficiency of the models in the two scenarios. Both charts report the total runtime (in log scale) on the X-axis and the MAE or the NDCG score on the Y-axis. The MAE refers to WikiTables' results. *Armadillo-inference* refers to Armadillo's runtime when the table embeddings are already available.

Tables 2 and 3 summarize the results obtained by Armadillo and the baselines in the task of table matching. The first one shows the performance of the TRL-based baselines, while the second one focuses on the non-TRL ones. Table 4 reports the performance of all models in the table querying scenario. Finally, Figure 4 provides a visualization of the tradeoff between effectiveness and efficiency of Armadillo and the baselines in both scenarios. In the following sections, we will discuss the results in detail.

4.1 Effectiveness

We analyze *how well* Armadillo performs on the task of predicting the overlap ratio between tables, in comparison to the ground truth, by analyzing its performances when performing the tasks of *table matching*, in Section 4.1.1, and *table querying*, in Section 4.1.2.

4.1.1 Table matching. To understand how well Armadillo and its competitors approximate the overlap ratio between pairs of tables, we report the mean absolute error (MAE) of their predictions computed using as ground truth the test sets of the GitTables and WikiTables triple datasets. To analyze the performance of Armadillo and its TRL competitors on different domains, we also perform a *cross-domain evaluation* by reporting the MAE when the models are trained on GitTables and evaluated on WikiTables and vice versa. In Table 2, the columns MAE Git-Git and MAE Wiki-Wiki refer to the MAE of the models trained on GitTables and evaluated on GitTables and trained on WikiTables and evaluated on WikiTables respectively, while MAE Git-Wiki MAE Wiki-Git refer to the cross-domain evaluation, i.e., trained on GitTables and evaluated on WikiTables and trained on WikiTables and evaluated on GitTables respectively. Figure 5 contains five box plots that show how the absolute error (AE) of Armadillo and its best-performing competitors varies across the

Table 2. Experimental results for the TRL models introduced in Section 3.3 in the table matching scenario. In the column names, the suffixes Git and Wiki identify the dataset that the column refers to, while Git-Git, Git-Wiki, Wiki-Wiki, and Wiki-Git refer to models trained on GitTables and evaluated on GitTables, trained on GitTables and Evaluated on WikiTables, and so on. The total time refers to the sum of the embedding time and the inference time.

Model	Total Time Git	Total Time Wiki	Embedding Time Git	Embedding Time Wiki	Inference Time Git	Inference Time Wiki	MAE Git Git	MAE Git Wiki	MAE Wiki Wiki	MAE Wiki Git
Sloth	80h:03m	15h:15m	0s	0s	80h:03m	15h:15m	0.000	0.000	0.000	0.000
EmbDI-T	-	19h:33m	-	19h:33m	-	10s	-	-	0.339	-
TURL-T	135h:33m	58m:51s	135h:23m	53m:00s	9m:48s	5m:51s	0.326	0.367	0.199	0.388
BERT-R	52h:47m	2h:32m	52h:36m	2h:32m	18s	11s	0.240	0.363	0.115	0.395
BERT-T	5h:32m	26m:11s	5h:21m	26m:00s	18s	11s	0.113	0.295	0.092	0.294
BERT-T-N	42h:25m	2h:30m	42h:15m	2h:30m	18s	11s	0.172	0.354	0.208	0.284
RoBERTa-R	54h:34m	2h:34m	54h:23m	2h:34m	18s	11s	0.197	0.349	0.144	0.392
RoBERTa-T	3h:45m	26m:11s	3h:34m	26m:00s	18s	11s	0.094	0.309	0.096	0.298
RoBERTa-T-N	6h:34m	37m:11s	6h:24m	37m:00s	18s	11s	0.177	0.417	0.190	0.428
Armadillo	**0h:28m**	**1m:37s**	**0h:28m**	**1m:30s**	**11s**	**7s**	**0.064**	**0.214**	**0.068**	**0.246**

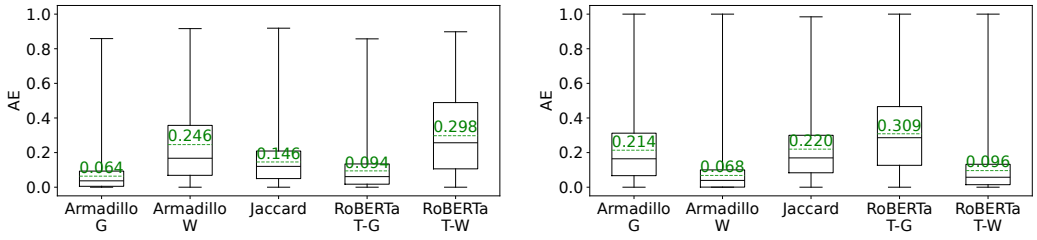
Table 3. Experimental results for the non-TRL models introduced in Section 3.3 in the table matching scenario.

Model	Total time-Git	Total time-Wiki	MAE-Git	MAE-Wiki
Sloth	80h:03m	15h:15m	0.000	0.000
Jaccard	**29s**	**6s**	0.146	0.220
Jaccard-B-U	1h:26m	11.85s	0.163	0.230
Jaccard-B-S	1h:26m	11.85s	0.165	0.270
Armadillo	0h:28m	1m:37s	**0.064**	**0.068**

Table 4. Experimental results for the models introduced in Section 3.3 in the table querying scenario.

Model	Total time	Inference time	NDCG@10	NDCG@1000
Sloth	171h:10m	171h:10m	1.00	1.00
Jaccard	**3m:56s**	3m:56s	**0.81**	0.71
Jaccard-B-U	6h:23m	6h:23m	**0.81**	0.70
Jaccard-B-S	6h:23m	6h:23m	0.78	**0.80**
TURL-T	6h:6m	1h:36m	0.77	0.65
BERT-R	4h:23m	2m:02s	0.77	0.64
BERT-T	1h:58m	2m:05s	0.77	0.68
BERT-T-N	3h:26m	2m:06s	0.66	0.61
RoBERTa-R	4h:25m	2m:05s	0.77	0.62
RoBERTa-T	1h:51m	2m:05s	0.77	0.65
RoBERTa-T-N	1h:57m	2m:05s	0.66	0.57
Armadillo	8m:59s	**1m:54s**	0.73	0.64

datasets, there, the MAE is indicated with a green dotted line and its value is reported above it. The upper chart in Figure 4 reports with a dashed horizontal line the MAE of Armadillo, showing intuitively that our model provides a lower MAE than the competitors.



(a) Absolute error on GitTables.

(b) Absolute error on WikiTables.

Fig. 5. Absolute error distribution of Armadillo, Jaccard similarity, and RoBERTa-T in approximating the overlap ratio calculated by Sloth [54]. The mean absolute error of the models is reported in green. The letters G and W identify the training dataset of the model.

On WikiTables we observe that Armadillo, in its domain-native version, obtains a MAE of 0.068, which is the best result. BERT-T and RoBERTa-T have a MAE of 0.092 and 0.096 respectively, performing not only better than BERT-R and RoBERTa-R, which have a MAE of 0.115 and 0.144, but also better than more resource-demanding models, i.e., EmbDI-T and TURL-T, which have a MAE of 0.339 and 0.199. A limitation of the BERT family models is that when working with noised data their performances drop significantly; indeed, BERT-T-N and RoBERTa-T-N have a MAE of 0.208 and 0.190. An explanation for this behavior is that their embeddings' quality relies on their knowledge of the table domain, e.g., Wikipedia is one of the training sources of BERT.

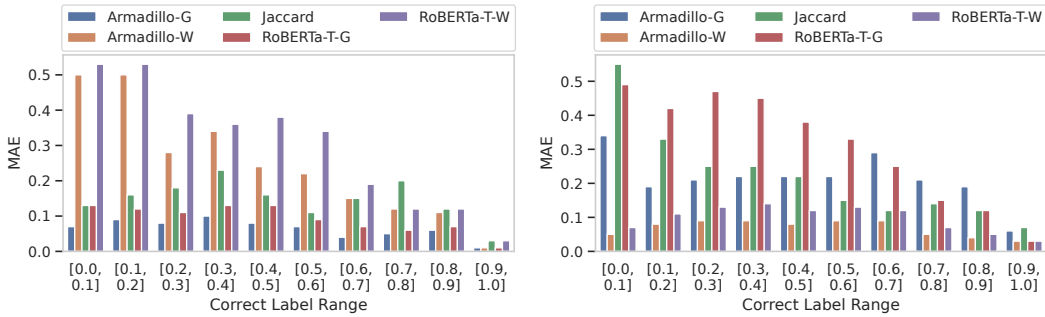
Regarding the non-TRL competitors, the Jaccard similarity produces the best results, with a MAE of 0.220 compared to a value of 0.230 for the Jaccard-B-U, and a MAE of 0.270 for the Jaccard-B-S.

The cross-domain evaluation highlights the limitations of Armadillo's architecture and the TRL competitors in general when used in a domain drastically different from the training one. The version of Armadillo trained on GitTables has a MAE of 0.214, which is similar to the MAE of the Jaccard similarity, while BERT-T and RoBERTa-T have a MAE of 0.295 and 0.309 and all the other TRL models have a MAE greater than 0.349.

On GitTables, the domain-native version of Armadillo is still the best with a MAE of 0.064, while the performances of BERT-R, TURL-T, RoBERTa-R, and BERT-T decrease, with MAEs of 0.240, 0.326, 0.197, and 0.113. On the other end, the performances of the Jaccard similarity, Jaccard-B-U, Jaccard-B-S, and RoBERTa-T, get better, with MAEs of 0.146, 0.163, 0.165, and 0.094.

The cross-domain evaluation has a different outcome, indeed, the versions of Armadillo, BERT-T, and RoBERTa-T trained on WikiTables have MAEs of 0.246, 0.294, and 0.298, performing worse than the Jaccard similarity and, for the model of the BERT family, comparably with BERT-T-N. An explanation for this result is that since the GitTables triple dataset contains a larger number of distinct tables than the WikiTables triple dataset and its tables are larger in size, the models are able to learn more generic information that proves useful in the cross-domain scenario. The other TRL models all have an MAE close to 0.4.

In Figure 6 we analyze how the MAE varies across pairs of tables with different overlap ratios. To carry out this evaluation we divided the samples in the test sets into 10 equally populated bins and measured the MAE of each bin. Figure 6a reports such results for GitTables and considers bins of 10k pairs, while Figure 6b does the same for WikiTables using bins of 6k pairs.



(a) MAE per overlap interval on GitTables.

(b) MAE per overlap interval on WikiTables.

Fig. 6. Distribution of Armadillo's and related baselines' mean absolute error (MAE) across pairs of tables with different overlap ratios. The letters G and W identify the training dataset of the model.

Analyzing the results on the two datasets, we can observe that the native version of Armadillo's error distribution is rather uniform across the bins, having similar performance regardless of the characteristics of the table pairs, which can have a low overlap in the cells as well as a high overlap. The only exception is the last bin, which contains highly overlapping table pairs. There the results are more precise for all the models, highlighting a bias in predicting very high overlaps, indeed, as shown in Figure 5, in very few cases all of the models get the results completely wrong, predicting a high overlap where there is almost none. RoBERTa-T follows a trend similar to Armadillo in both of the datasets, while we can observe that the Jaccard similarity, especially on WikiTables, tends to overestimate the overlaps.

In the cross-domain scenario, the Armadillo and RoBERTa-T versions trained on the WikiTables and evaluated on GitTables tend to overestimate the overlap. On WikiTables the results are slightly different, indeed, Armadillo performs better than the Jaccard similarity and has a more uniform MAE distribution between the bins. RoBERTa-T, instead, follows a trend similar to the Jaccard.

Regarding the relationship between the MAE of Armadillo and the areas of the tables, similarly to what we did for the true overlap ratio, we grouped the samples in the test sets in 10 bins by area ratio, i.e., the ratio between the area of the smaller table in the pair and the larger one. We observed that in both datasets Armadillo tends to perform better for high values of area ratio, i.e., when the tables in the pairs have similar sizes. This result is justified by the area ratio distribution of the pairs in the test datasets, indeed, more than half of them have an area ratio between 0.9 and 1.

As a final note, although rather variable errors exist for all approaches, as can be seen in Figure 5, Armadillo produces the smallest interquartile variance, demonstrating its robustness to different table characteristics.

4.1.2 Table querying. In this section, we analyze how well Armadillo performs when compared with the baseline approaches in a table querying scenario, i.e., finding the top k most similar tables in a collection with respect to a user query. In our experiments, we used a query set composed of 100 tables and a collection of 10k tables extracted from GitTables, none of these tables were used during the training and validation of the models.

To measure the performances, we used the NDCG (Normalized Discounted Cumulative Gain) score [28], a metric with values between 0 and 1 that measures how well a recommendation system is ranking the top k elements of a set with respect to the ideal ranking.

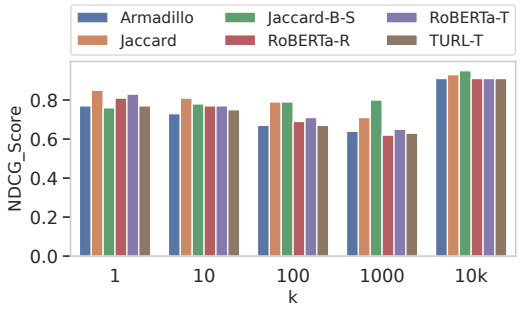


Fig. 7. NDCG@k score for increasing values of k. The TRL models were trained on GitTables.

In Figure 7, we show the results of this experiment by visualizing values of k that are powers of ten, i.e., 10^0,10^1,10^2,10^3, and 10^4. In Table 4, we report the values obtained with $k = 10^1$ and $k = 10^3$.

Both Figure 7 and Table 4 show that the TRL models have worse performances than the Jaccard-based ones, indeed, even though these methods are not aware of the structure of the tables, many cases of highly overlapping tables are just duplicate tables or table containments, and for both these categories of relationships, the structure awareness is less relevant than for smaller values of overlap. Furthermore, TRL models are trained to minimize the AE error on a single prediction, and even though small overlap differences between different pairs are important in a ranking scenario, they may be overlooked during training in favor of minimizing the larger ones.

In the table reclamation scenario, we measured the average of reduction ratio (RR) and pair completeness (PC) [37] across all the set of candidate generating tables obtained filtering out tables with an overlap ratio with the query table lower than a specified threshold. We observed that the best results are obtained using thresholds of 0.1 and 0.2, which have RRs of 0.54 and 0.77 and PCs of 0.71 and 0.50 respectively. More extreme results are obtained by using thresholds of 0.01 and 0.4, with RRs of 0.31 and 0.98 and PCs of 0.93 and 0.26.

4.2 Efficiency

We analyze *how fast* Armadillo is in the task of predicting the overlap ratio between tables by analyzing its performances when performing the tasks of *table matching*, in Section 4.2.1, and *table querying*, in Section 4.2.2.

4.2.1 Table matching. In this section, we provide evaluations that are complementary to the ones of Section 4.1.1 by analyzing the efficiency of Armadillo in calculating the overlap ratio between pairs of tables. In Table 2, the fourth and fifth columns report the time needed by Armadillo and its TRL competitors to embed all the tables that appear in the test set of the GitTables and WikiTables triple datasets. The sixth and the seventh columns report the time needed by the TRL models to compute the overlap ratio when the embeddings are already available, i.e., their inference time. Finally, the second and third columns report the Total time required by the models to compute the overlaps, i.e., the sum of their embedding and inference time. Table 3 reports the results for the non-TRL competitors, since they have no embedding time, it contains only the total time.

As also highlighted by the vertical dotted lines in the topmost chart of Figure 4, on both datasets, Armadillo is considerably faster than the TRL competitors, needing less than thirty minutes for GitTables and less than two minutes for WikiTables. Both BERT-T and RoBERTa-T, the fastest of the baselines, needed more than three hours for embedding GitTables and almost thirty minutes for WikiTables, their noised versions, i.e., BERT-T-N and RoBERTa-T-N, probably due to the different

tokenization of the noised cell values, were slower, needing more than two hours and a bit less than one hour to embed the tables of WikiTables. BERT-R and RoBERTa-R needed more than fifty hours for GitTables and more than two and a half hours for WikiTables. TURL-T, even though it needed less than one hour to process WikiTables, proved not to scale well with large table sizes and needed more than five days for GitTables. Finally, EmbDI-T needed almost one day to generate the embeddings of WikiTables, which is more than the time needed by Sloth to compute all of the overlaps from scratch.

The fifth and sixth columns of Table 2 report the inference time needed by Armadillo and its competitors to compute the actual table overlaps. For Armadillo, this time refers to the computation of the cosine similarity and requires less than seven seconds for WikiTables and less than twelve for GitTables. For the other TRL models, the inference phase requires *fine tuning* the table embeddings through the scaling head and computing their cosine similarity, this operation required around ten seconds on WikiTables and slightly less than twenty seconds on GitTables for each one of the TRL competitors. The only exception is TURL-T, which required additional operations for the embedding fine tuning for a total of around five minutes on WikiTables and ten minutes on GitTables. Regarding the non-TRL competitors, they compute the overlaps from scratch and have different inference times. Sloth needed a bit more than fifteen hours for WikiTables and more than 80 hours for GitTables, while Jaccard-B-U and Jaccard-B-S needed less than twelve seconds and one and a half hours for WikiTables and GitTables respectively. Finally, the standard Jaccard similarity, needing less than thirty seconds to process GitTables and around six for WikiTables, has the lower total time of all the models, which is also comparable to Armadillo's inference time.

Figure 8 presents the results of a workload test to evaluate the cumulative runtime of the models while predicting the overlap ratios of the pairs in the test set of WikiTables. The x-axis reports the number of overlaps computed and the y-axis, the current cumulative runtime. The chart contains one line for each model and since there are a few models considerably slower than others, it contains in the top left corner a *zoom* on the fastest models to help the visualization. This test represents a sort of *online* usage of the models where it is not possible to apply any optimization, e.g., exploiting pre-computed table embeddings or computing more overlaps in parallel, where all the tables are assumed to be new, i.e., they have never been encountered before and for each pair, the models have to find new table embeddings from scratch and compute their similarity.

At first impact, it is clear that all of the models except EmbDI are several times faster than Sloth, the black line. A second consideration is that the lines do not intersect, i.e., the faster models' cumulative runtime is consistently lower than Sloth's. Looking into the zoomed area, RoBERTa-R is the slowest of the faster models, followed by TURL. Then, RoBERTa-T, and finally, Armadillo and the Jaccard similarity, which we recall needed less than one minute to compute all of the overlaps.

As a final note, as shown in Table 2, Armadillo spends most of the time computing table embeddings and, more specifically, constructing the intermediate graph-based representation of the tables. We further inspected this behavior and identified a correlation between the table area and the total embedding time, suggesting that larger tables have greater embedding time.

4.2.2 Table querying. In this section, we provide evaluations that are complementary to the ones of Section 4.1.2 by analyzing the efficiency of Armadillo in a table querying scenario. To compute the 1 000 000 overlaps for this experiment, Sloth needed 7 days of machine time. Overall, Armadillo took approximately ten minutes to embed the 10 100 tables needed for the experiment and to compute the 1 000 000 cosine similarities between the pairs. BERT-T, RoBERTa-T, and RoBERTa-T-N needed around two hours for the same operation, BERT-R, BERT-T-N, and RoBERTa-R around four hours, and TURL, Jaccard-B-U, and Jaccard-B-S around six hours. Finally, the Jaccard similarity was the fastest method and needed four minutes in total. A similar trend can be observed in the

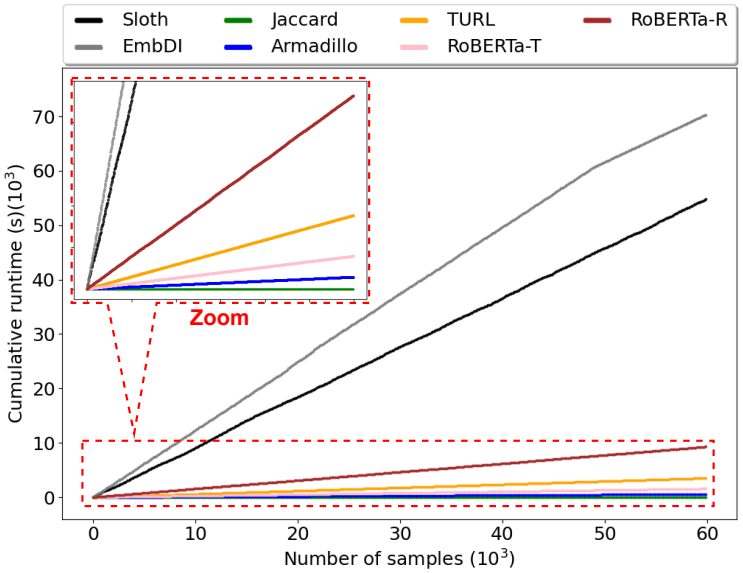


Fig. 8. Cumulative runtime of the models while computing the overlap ratio of the 60k test pairs of WikiTables.

table reclamation scenario, where Armadillo computed almost three million overlaps and half overlaps in less than twenty minutes.

In any case, also in this task, Armadillo is the best TRL approach in terms of efficiency and the best in general, if we consider only its inference time, and even though its effectiveness is not as good, it was not trained for a ranking task, and developing an architecture that is specific for it may be relevant future work.

4.3 Architectural validation

To validate our architecture and justify our design choices we performed (1) an ablation study to measure the impact of variations on the core components and (2) a performance stability study to show that the model's performs consistently across different data from the same domain.

Ablation study. The three core components of the Armadillo architecture proposed in Section 2 are *node embedding initialization*, *node embedding aggregation*, and *overlap ratio computation*. We separately measured the impact of changes on these components and summarized an overview of our results in Table 5.

Instead of initializing the node embeddings using the SHA-256 algorithm, we tried a BERT [11] based approach, a Fasttext [7] based approach, and a random initialization. Our experiments suggest that a random initialization, despite being slightly faster than SHA-256, is unable to provide information about the content of the nodes, scoring a MAE of 0.1792 and delivering the worst performance. The MAE obtained using Fasttext and BERT are higher than the ones obtained using SHA-256 as well, suggesting that hash-values contained all the structural information needed by the model for the overlap computation and that during the embedding process some of this information is lost. Furthermore, runtime-wise they are outperformed, building the table graph almost three and twenty times slower than Armadillo's configuration.

Instead of computing the final table embedding by averaging all the column, row, and cell embeddings, we also tried aggregating their combinations: only cell, row, or column embeddings,

Table 5. Ablation study comparing alternative component implementations to the Armadillo configuration.

Component	Configuration	Total time	MAE
Initialization	Random	3m:2s	0.1792
	BERT	30m:16s	0.0736
	Fasttext	6m:31s	0.0917
Aggregation	Rows	3m:21s	0.0727
	Columns	3m:25s	0.0725
	Cells	3m:52s	0.0731
	Rows-Columns	3m:27s	0.0745
	Rows-Cells	3m:22s	0.0708
	Columns-Cells	3m:7s	0.0692
Similarity	Absolute value	3m:25s	0.0709
Armadillo		3m:23s	**0.0680**

only row and column embeddings, only row and cell embeddings, and only column and cell embeddings. Although our experiments show that Armadillo's configuration delivers the best performance, they are comparable with the ones of the other models, especially the ones obtained by aggregating cell and column nodes, suggesting that their embeddings contain the most relevant information for overlap computation. Indeed, even though the row embeddings are not directly used, as shown in Section 2.2, they are part of the receptive field of column and cell nodes and their embeddings entail information about them as well.

Finally, instead of computing the overlap ratio of two tables as the maximum value between zero and the cosine similarity between their embeddings, we tried using the absolute value of the cosine similarity. Our experiments suggest that the two approaches perform similarly, but Armadillo's one provides a lower MAE.

Performance stability. To evaluate the consistency of Armadillo's performance across different data from the same domain, we swapped the test and validation sets of the WikiTables triple dataset and used them to train another version of Armadillo. We performed the same operation to train a new version of Armadillo on GitTables. The models obtained a MAE of 0.069 both on WikiTables and GitTables, these results are close to the values obtained by the versions of Armadillo trained using test and validation sets in the *normal* ordering, which are 0.068 and 0.064 respectively.

5 Related work

In this section, we review the research areas that are most connected to our work: (1) discovery and integration of related tables, (2) table representation learning, and (3) graph neural networks.

5.1 Discovery and integration of related tables

A line of work in the literature focuses on managing tabular data in data lakes. In this domain, we identify three main tasks: discovering related tables, integrating them, and detecting duplicates.

For related table discovery, many approaches identify joinable tables to extend schemas by finding shared join attributes using exact or fuzzy matching [12, 14, 45]. Zhu et al. [57] propose JOSIE, which frames joinable table discovery as an overlap set similarity search problem, offering scalability for large datasets. Other approaches focus on discovering unionable tables, i.e., tables that can be merged to increase their number of tuples [17]. Nargesian et al. [40] approach the problem by identifying unionable attributes, i.e., attributes that derive from the same domain and that have significant syntactic and semantic similarity. SANTOS [29], further extends this approach

by analyzing relationships between columns. More recently, Fan et al. [18] introduced Starmie which leverages pre-trained language models to encode advanced column semantics using full table context. There is a line of work that further generalizes this concept by discovering related tables in a broad sense. Sarma et al. [43] provide a framework for finding bondings between tables, including joinable and unionable tables, with scalable algorithms capable of handling over a *million tables*. Other cases of related table discovery focus on identifying similar datasets in data lakes to support data wrangling [6] or exploring related tables for data science pipelines, such as retrieving training data and features in environments like Jupyter Notebooks [56].

The discovery phase is followed by the integration process, which aims to generate coherent tabular data by combining related tables. Khatiwada et al. [30] propose ALITE, a framework that exploits the full disjunction operator to handle cyclic join patterns over incomplete tabular data. To aid the information integration process, Saha et al. [42] introduce schema covering, where a complex schema is aligned with a set of concepts to identify correspondences, i.e., a cover of the schema. Gal et al. [19] study the tradeoff between completeness, i.e., the part of the schema that can be covered by the set of concepts in the cover, and ambiguity, i.e., the amount of overlap between concepts in the cover. Fan et al. [16] propose Gen-T, a framework for table reclamation that aims to identify a set of source tables to closely reconstruct a given table by performing both data discovery and integration. The authors also apply Gen-T to validate tabular data generated by large language models (LLMs) [15]. In our experiments, we explored Armadillo's capabilities as blocking method for the task of table reclamation.

The third category involves detecting duplicate tables. Bleifuß et al. [4] identify duplicate tables in adjacent Wikipedia snapshots to track the evolution of tables over time. Koch et al. [32] focus on table matching in data lakes, distinguishing between duplicate table retrieval (querying a table against a collection) and lake de-duplication (comparing all tables), but considering only exact matches and containment, hence simplified scenarios compared to those identifiable through the table overlap.

5.2 Table representation learning

We call *tabular elements* components of tables such as rows, columns, cells, and captions. Two main categories of approaches exist for generating embeddings of tabular elements: transformer-based models and graph-based models. Transformer-based approaches flatten the two-dimensional table into a textual format [1], often by serializing rows or columns and inserting separators and data types [10, 13, 24, 44, 52]. Structural awareness is added through positional embeddings [24, 26, 52], modified attention mechanisms (e.g., vertical attention in TaBERT [52] and combined vertical-horizontal attention in [46]), or pretraining tasks specific to tabular data, such as masking cells, columns, or entities [10, 24, 52].

Graph-based approaches, on the other hand, represent tables as graphs, where nodes correspond to tabular elements and edges to relationships. EmbDI [8] represents tables as tripartite graphs connecting column, token, and record nodes, and generates embeddings via a node2vec-style approach [21]. Similar graphs [20, 34] are also exploited to generate meaningful attribute and record representations that are used to support the entity resolution task. In MGNETS [9] one or more graphs are used to model tables' relations and to support the table search task. GTR [48] instead feeds a graph transformer [53] with a tabular graph to solve the table retrieval task. Finally, other approaches, such as Retro [22], generate relational embeddings by refining pre-trained word embeddings using foreign key relationships. Since it creates an intermediate graph-based representation of the table which is processed by a graph neural network model, Armadillo can be classified in this category of approaches.

5.3 Graph neural networks

Graph neural networks are a category of deep learning models that are designed to process graphs. Kipf et al. [31] proposed one of the first architectures in this category, GCN (graph convolutional network), which adopts a scalable semi-supervised approach to learn node embeddings that encode both local graph structure and features of nodes. GraphSAGE, proposed by Hamilton et al. [23], further extends the architecture by learning a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. While in previous approaches neighboring nodes contribute equally to the embedding of a target node, in graph attention networks (GATs) [47] each node can influence the embeddings of its neighbors differently. This is achieved by using an attention mechanism whose weights are learned directly by the GNN to discriminate the more relevant neighbors from the less important ones. Finally, another popular GNN model is GIN (graph isomorphism network) [51], which generates graph representations that encode isomorphism relationships between graphs.

6 Conclusion and future work

In this paper, we presented Armadillo, an approach to efficiently determine the overlap ratio between two tables. Its main applications are in the discovery of duplicate tables, multiple versions of the same table, and related tables in general. Armadillo addresses this problem by generating table embeddings that encode overlap relationships between tables in their similarities and that can be efficiently compared. More specifically, Armadillo first creates intermediate representations of tables based on tripartite graphs that embed knowledge about the structure and content of the table. These graphs are then provided as input to a graph neural network that learns graph embeddings. Finally, the embeddings' cosine similarity approximates the overlap ratio between the input tables.

We also introduced two new datasets derived from GitTables and WikiTables, which contain pairs of tables labeled with their overlap. After analyzing the performance of Armadillo on these datasets, we observed that it can efficiently compute the table overlap ratio, committing an absolute error lower than seven percent on average.

Regarding future work, we plan to improve the cross-domain performance of Armadillo by exploiting *domain adaptation* techniques and to further extend Armadillo by training a new model specifically for a table-ranking scenario. Then, we plan to relax the original definition of table overlap, which relies on exact matching, by exploiting a more approximate fuzzy matching logic to identify overlapping tables even with noisy data.

Acknoledgements

This work was partially funded by SAP, by MUR within the PRIN "Discount Quality" project (code 202248FWFS), and by Cineca within the ISCRA-C program (code HP10CSFAPN).

References

[1] 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.

[2] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *ISWC (1) (Lecture Notes in Computer Science, Vol. 9366)*. Springer, 425–441.

[3] Tobias Bleifuß, Leon Bornemann, Theodore Johnson, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. 2018. Exploring Change: A New Dimension of Data Analytics. *Proceedings of the VLDB Endowment (PVLDB)* 12, 2 (2018), 85–98. https://doi.org/10.14778/3282495.3282496

[4] Tobias Bleifuß, Leon Bornemann, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. 2021. Structured Object Matching across Web Page Revisions. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 1284–1295. https://doi.org/10.1109/ICDE51399.2021.00115

[5] Tobias Bleifuß, Leon Bornemann, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. 2021. The Secret Life of Wikipedia Tables. In *Proceedings of the Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores (SEA Data @ VLDB) (CEUR Workshop Proceedings, 2929)*. 20–26. https://ceur-ws.org/Vol-2929/paper4.pdf

[6] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, 709–720. https://doi.org/10.1109/ICDE48307.2020.00067

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2017. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics* 5 (2017), 135–146.

[8] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 1335–1349. https://doi.org/10.1145/3318464.3389742

[9] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Dawei Yin, and Brian D. Davison. 2021. MGNETS: Multi-Graph Neural Networks for Table Search. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*. ACM, 2945–2949.

[10] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. TURL: Table Understanding through Representation Learning. *SIGMOD Record* 51, 1 (2022), 33–40. https://doi.org/10.1145/3542700.3542709

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1)*. Association for Computational Linguistics, 4171–4186.

[12] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2023. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. *PVLDB* 16, 10 (2023), 2458–2470.

[13] Julian Martin Eisenschlos, Maharshi Gor, Thomas Müller, and William W. Cohen. 2021. MATE: Multi-view Attention for Table Transformer Efficiency. In *EMNLP (1)*. Association for Computational Linguistics, 7606–7619.

[14] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2022. MATE: Multi-Attribute Table Extraction. *PVLDB* 15, 8 (2022), 1684–1696.

[15] Grace Fan, Roee Shraga, and Renée J. Miller. 2024. Finding Support for Tabular LLM Outputs. In *VLDB Workshops*. VLDB.org.

[16] Grace Fan, Roee Shraga, and Renée J. Miller. 2024. Gen-T: Table Reclamation in Data Lakes. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, 3532–3545.

[17] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. 2023. Table Discovery in Data Lakes: State-of-the-art and Future Directions. In *SIGMOD Conference Companion*. ACM, 69–75.

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

[19] Avigdor Gal, Michael Katz, Tomer Sagi, Matthias Weidlich, Karl Aberer, Nguyen Quoc Viet Hung, Zoltán Miklós, Eliezer Levy, and Victor Shafran. 2013. Completeness and Ambiguity of Schema Cover. In *OTM Conferences (Lecture Notes in Computer Science, Vol. 8185)*. Springer, 241–258.

[20] Congcong Ge, Pengfei Wang, Lu Chen, Xiaoze Liu, Baihua Zheng, and Yunjun Gao. 2023. CollaborEM: A Self-Supervised Entity Matching Framework Using Multi-Features Collaboration. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 35, 12 (2023), 12139–12152.

[21] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. ACM, 855–864. https://doi.org/10.1145/2939672.2939754

[22] Michael Günther, Philipp Oehme, Maik Thiele, and Wolfgang Lehner. 2020. Learning from Textual Data in Database Systems. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*. ACM, 375–384.

[23] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 1024–1034. https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html

[24] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *ACL*. Association for Computational Linguistics, 4320–4333.

[25] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. *Proc. ACM Manag. Data* 1, 1 (2023), 30:1–30:17.

[26] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *NAACL-HLT*. Association for Computational Linguistics, 3446–3456.

[27] Jaccard. 1912. The distribution of the flora of the alpine zone. In *New Phytologist* (Cargse, France.), Vol. 11. 37–50.

[28] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* 20, 4 (2002), 422–446.
[29] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1 (2023), 9:1–9:25.
[30] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J. Miller. 2022. Integrating Data Lake Tables. *PVLDB* 16, 4 (2022), 932–945.
[31] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR (Poster)*. OpenReview.net.
[32] Maximilian Koch, Mahdi Esmailoghli, Sören Auer, and Ziawasch Abedjan. 2023. Duplicate Table Discovery with Xash. In *Proceedings of the Conference Datenbanksysteme in Business, Technologie und Web Technik (BTW) (LNI, Vol. P-331)*. Gesellschaft für Informatik e.V., 367–390.
[33] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of Massive Datasets*. http://www.mmds.org
[34] Bing Li, Wei Wang, Yifang Sun, Linhan Zhang, Muhammad Asif Ali, and Yi Wang. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 8172–8179.
[35] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. 2012. Truth Finding on the Deep Web: Is the Problem Solved? *Proceedings of the VLDB Endowment (PVLDB)* 6, 2 (2012), 97–108. https://doi.org/10.14778/2535568.2448943
[36] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019).
[37] Matthew Michelson and Craig A. Knoblock. 2006. Learning Blocking Schemes for Record Linkage. In *AAAI*. AAAI Press, 440–445.
[38] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR (Workshop Poster)*.
[39] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *PVLDB* 12, 12 (2019), 1986–1989. https://doi.org/10.14778/3352063.3352116
[40] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *PVLDB* 11, 7 (2018), 813–825.
[41] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. 2023. A Survey on Oversmoothing in Graph Neural Networks. *CoRR* abs/2303.10993 (2023).
[42] Barna Saha, Ioana Stanoi, and Kenneth L. Clarkson. 2010. Schema covering: a step towards enabling reuse in information integration. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 285–296.
[43] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 817–828. https://doi.org/10.1145/2213836.2213962
[44] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çagatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-trained Language Models. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 1493–1503.
[45] Sahaana Suri, Ihab F. Ilyas, Christopher Ré, and Theodoros Rekatsinas. 2021. Ember: No-Code Context Enrichment via Similarity-Based Keyless Joins. *PVLDB* 15, 3 (2021), 699–712.
[46] Mohamed Trabelsi, Zhiyu Chen, Shuo Zhang, Brian D. Davison, and Jeff Heflin. 2022. StruBERT: Structure-aware BERT for Table Search and Matching. In *WWW*. ACM, 442–451.
[47] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR (Poster)*. OpenReview.net.
[48] Fei Wang, Kexuan Sun, Muhao Chen, Jay Pujara, and Pedro A. Szekely. 2021. Retrieving Complex Tables with Multi-Granular Graph Representation Learning. In *Proceedings of the International Conference on Information Retrieval (SIGIR)*. ACM, 1472–1482.
[49] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016).
[50] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=ryGs6iA5Km

[51] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*. OpenReview.net.
[52] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*. Association for Computational Linguistics, 8413–8426.
[53] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. 2019. Graph Transformer Networks. In *NeurIPS*. 11960–11970.
[54] Luca Zecchini, Tobias Bleifuß, Giovanni Simonini, Sonia Bergamaschi, and Felix Naumann. 2024. Determining the Largest Overlap between Tables. *Proceedings of the ACM on Management of Data (PACMMOD)* 2, 1 (2024), 48:1–48:26.
[55] Li Zhang, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *Proceedings of the International Conference on Information Retrieval (SIGIR)*. ACM, 1029–1032. https://doi.org/10.1145/3331184.3331333
[56] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 1951–1966.
[57] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 847–864.

Received October 2024; revised January 2025; accepted February 2025