DeepJoin: Joinable Table Discovery with Pre-trained Language Models

[1]Yuyang Dong, [2,3]Chuan Xiao, [1]Takuma Nozawa, [1]Masafumi Enomoto, [1]Masafumi Oyamada
[1]NEC Corporation, [2]Osaka University, [3]Nagoya University
{dongyuyang,nozawa-takuma,masafumi-enomoto,oyamada}@nec.com,chuanx@ist.osaka-u.ac.jp

ABSTRACT

Due to the usefulness in data enrichment for various machine learning tasks, joinable table discovery has become an important operation in data lake management. Existing approaches either target equi-joins, the most common way of combining tables for create a unified view, or semantic joins, which tolerate misspellings and different formats to deliver more join results. However, they are mainly exact solutions whose running time is linear in the size of query column and and target table repository. In this paper, we propose DeepJoin, a deep learning model for efficient joinable table discovery. Our solution is essentially an embedding-based retrieval, which employs a pre-trained language model (PLM) and is designed in a fashion that serves both equi- and semantic joins. We propose a set of contextualization options to transform column contents to a sequence fed to the PLM, which produces a column embedding such that columns are expected to be joinable if they are close in the vector space. Due to the fixed length of column embedding, the subsequent index lookup and search becomes independent of the query size. By utilizting approximate nearest neighbor search, the search time is rendered subliner in the repository size. To train the model, we design a series of techniques for generating positive and negative examples, as well as data augmentation. The experiments on two real datasets demonstrate that by training on a small subset, DeepJoin generallizes to large datasets and reports higher accracy than alternative approximate solutions. It is even more effective than the exact solution to semantic joins when evaluated on expert-labeled data. At the same time, DeepJoin equipped with CPU is 7 to 57 times faster than the exact solutions, and the speedup can be two-orders of magnitude when equipped with GPU.

PVLDB Reference Format:
[1]Yuyang Dong, [2,3]Chuan Xiao, [1]Takuma Nozawa, [1]Masafumi Enomoto, [1]Masafumi Oyamada. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

Join is a fundamental and essential operation that connects two or more tables. It is also a significant technique applied to relational database management systems and business intelligence tools for data analysis. The benefits of joining tables are not only in the database fields (e.g., data integration) but also in the machine learning (ML) fields such as feature augmentation and data enrichment.

[14] for data enrichment for ML.

With the trends of open data movements by governments and the dissemination of data lake solutions in industries, we are provided with more opportunities to obtain a huge number of tables from data lakes (e.g., WDC Web Table Corpus) and make use of them to enrich our local data. As such, a local table can be regarded as a query, and our request is to look for joinable tables in the data lake. Many researchers studied the problem of data discovery in data lakes. Early attempts studied the problem of finding joinable tables [56, 58] and focused on evaluating the joinability between columns by taking the overlap number of equi-joined records. A deeper view on the semantic level, such as utilizing word embeddings, enables us to identify text with the same or similar meanings, hence to tackle the data heterogenity. PEXESO [15] is a recent solution which finds semantically joinable tables. In PEXESO, each cell is transformed to a high-dimensional vector by a word embedding approach such as fastText [18]. A column is thus represented as a multiset of high-dimensional vectors. Then, it leverages the similarity between vectors to evaluate the joinability between columns. Whereas it is able to find more tables and its usefulness in enriching data for ML is demonstrated, it requires two user-specified thresholds to define matching cells and joinable columns. Moreover, to search for joinable columns, it relies on a pivot-based index, which tends to be less scalable for large-scale high-dimensional datasets.

Contributions. Our contributions are summarized as follows.

- We propose DeepJoin, a framework for joinable search discovery in data lakes. Our solution is able to detect both equi-joinable and semantically joinable tables. It is designed in a top-k fashion such that users are not required to specific any threshold.
- To find both equi-joinable and semantically joinable tables, we propose a column embedding approach which employs pre-trained language models (PLMs). We propose a prompt engineering method to transform table contents to text which is then fed to the PLM.
- Through metric learning, the PLM is able to embed columns to vectors such that columns are expected to be joinable if they are close in the vector space. In addition, we employ a state-of-the-art index for approximate nearest neighbor search to fast search speed.
- We propose a series of data augmentation techniques to cope with the case of limited labeled data for training.
- We conduct experiments on real datasets to demonstrate the effectiveness and the efficiency of DeepJoin in finding joinable tables.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

FIXME!!! We can mention that candidate generation + ranking may further improve the performance (in Section 1).

2 PRELIMINARIES

2.1 Problem Definition

Given a data lake of tables, we extract all the columns in these tables, except those never appear in a join predicate (e.g., BLOBs), and create a repository of tables, denoted by $\mathcal{X}$. Given a query column Q, our task is to search $\mathcal{X}$ and find the columns joinable to Q. In this paper, we target equi-joins and semantic joins. Equi-join is the most common way of combining tables for create a unified view [13] and can be easily implemented using SQL, while semantic join is defined over word embeddings and is able to handle data with misspellings and different formats/terminologies (e.g., "American Indian & Alaska Native" v.s. "Mainland Indigenous"), hence delivering more join results.

Given a query column Q and a target column target column X in $\mathcal{X}$, the joinability from Q to X is defined by the following equation.

$$jn(Q, X) = \frac{|Q_M|}{|Q|}, \tag{1}$$

where $|\cdot|$ measures the size (i.e., the number of cells) of a column, and Q_M is composed of the cells in Q that have a match in X. Here, the term "match" depends on the join type, i.e., an equi-join or a semantic join. We normalize the size of Q_M by the size of Q to return a value between 0 and 1. Moreover, the joinability is not always symmetric, depending on the definition of Q_M.

For equi-joins, we model each column as a *set* of cells by removing duplicate cell values, and define the equi-joinability as follows.

Definition 2.1 (Equi-Joinability). The equi-joinability from the query column Q to a target column X in $\mathcal{X}$ counts the intersection between Q and X, normalized by the size of Q; i.e., in Equation 1,

$$Q_M = Q \cap X. \tag{2}$$

The equi-joinability defined above counts the the distinct number of tuples in Q that join with X, and thus can be used to measure the equi-joinability [56]. Our method can be also extended to the case when columns are modeled as multisets, so as to support one-to-many, many-to-one, and many-to-many joins. In this case, we may measure the joinability by the number of join results are normalize it by the product of $|Q|$ and $|X|$, instead of $|Q|$ in Equation 1.

For semantic joins, we consider string columns and embed the value of each cell to a metric space $\mathcal{V}$ (e.g., word embedding by fastText [18]). As such, each string column is transformed to a multiset of vectors. We abuse the notation of a column to denote its multiset of vectors. Then, the notion of vector matching is defined as follows.

Definition 2.2 (Vector Matching). Given two vectors v_1 and v_2 in $\mathcal{V}$, a distance function d, and a threshold τ, v_1 **matches** v_2 if and only if the distance between v_1 and v_2 does not exceed τ. We use notation $M^d_\tau(v_1, v_2)$ to denote if v_1 matches v_2; i.e., $M^d_\tau(v_1, v_2) = 1$, iff. $d(v_1, v_2) \leq \tau$, or 0, otherwise.

Given a query column Q and a target column X, the semantic-joinability is defined using the number of matching vectors [1].

[1]Besides this definition, semantic joins are also investigated in [21], yet it studies the problem of performing joins rather than searching for joinable columns

Definition 2.3 (Semantic-Joinability). The semantic-joinability from Q to X counts the number of vectors in Q having at least one matching vector in X, normalized by the size of Q; i.e., in Equation 1,

$$Q_M = \{\, q \mid q \in Q \wedge \exists x \in X \text{ s.t. } M^d_\tau(q, x) = 1 \,\}. \tag{3}$$

An advantage of the above definitions is that for both equi- and semantic-joinability, the training data can be labeled by an exact algorithm (e.g., JOSIE [56] and PEXESO [15]) rather than experts, so that our model can be trained in a self-supervised manner. Following the above definitions, we model the problem of joinable table discover as the following top-k search problem, where joinability jn is defined using either Definition 2.1 or 2.3.

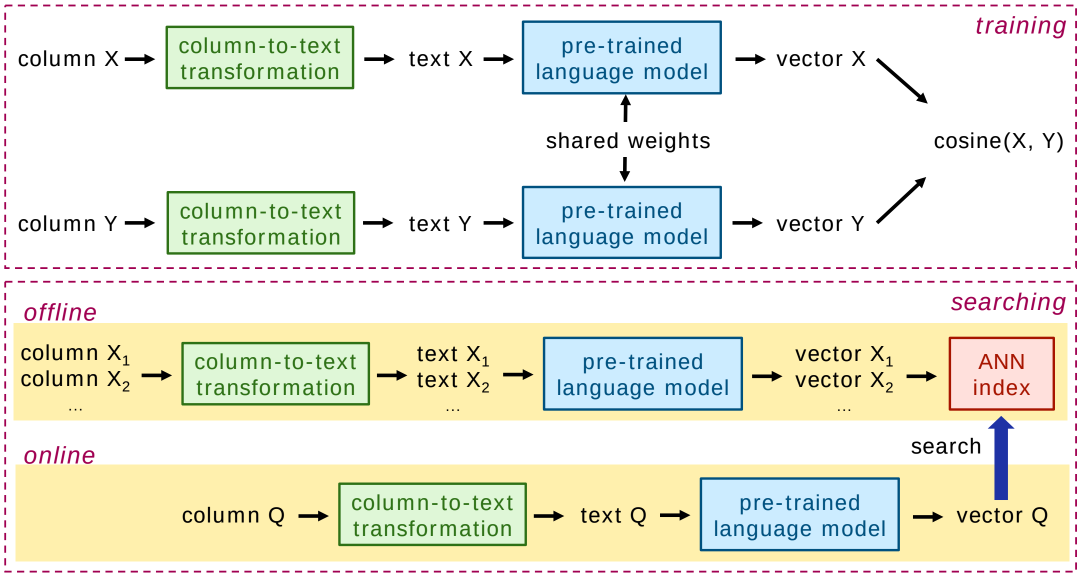
Problem 1 (Joinable Table Discovery). *Given a query column Q and a collection of columns $\mathcal{X}$, the joinable table discovery problem is to find the top-k columns of $\mathcal{X}$ with the largest joinability $jn(Q, X)$ to Q. Formally, we find a subset $\mathcal{R} \subseteq \mathcal{X}$, $|\mathcal{R}| = k$, and* $\min\{\, jn(Q, X) \mid X \in \mathcal{R}) \,\} \geq jn(Q, Y), \forall Y \in \mathcal{X} \backslash \mathcal{R}$.

2.2 State-of-the-Art

JOSIE [56], the state-of-the-art solution to the equi-joinable table discovery problem, regards the problem as a top-k set similarity search with overlap ($|Q \cap X|$) as similarity function, and builds its search algorithm upon prefix filter [7] and positional filter [51], which have been extensively used for solving set similarity queries. JOSIE creates an inverted index over $\mathcal{X}$, which which regards each cell value as a token and maps each token to a postings list of columns having the token. Then, it finds a set of candidate columns by retriving the postings lists for a subset of the tokens in Q (called prefix). Candidates are verified for joinability and the top-k is updated. While index access and candidate verification are processed in an alternative manner, JOSIE features the techniques to determine the their order, so as to optimize for large columns and large token universes, which are often observed in data lakes.

PEXESO [15] is an exact solution to semantic-joinable table discovery. It employs pivot-based filtering [8], which selects a set of vectors as pivots and pre-computes distances to these pivots for all the vecotrs in the columns of $\mathcal{X}$. Then given the vectors of the query Q, non-matching vectors are pruned by the triangle inequality. A hierarchical grid index is built to filter non-joinable columns when counting the number of matching vectors.

As for the weakness, JOSIE inherits the drawback of prefix filter, whose performance highly depends on the data distribution and yields a worst-case time complexity of $O(|\mathcal{X}| \cdot (|Q| + \overline{|X|}))$, where $\overline{|X|}$ stands for the average size of the columns in $\mathcal{X}$. For PEXESO, despite a claimed sublinear search time complexity of $O(\log |\mathcal{X}_V| \cdot \log |Q|)$, where $\mathcal{X}_V$ denotes the multiset of all the vectors in the target columns, it relies on a user-specific threshold for the count of matching vectors. This does not apply to the top-k case, and downgrades the algorithm to be linear in $|Q| \cdot |\mathcal{X}_V|$, because at the early stage of searching, due to the low count of matching vectors in the temporary top-k, there is almost no pruning. In general, for search time, both JOSIE and PEXESO are linear in column size and dataset size, which compromises their scalability to long columns and large datasets. On the other hand, it is unnecessary to always find an exact answer, because for data scientists are usually concerned with part of the top-k results and choose a subset of them



training
column X
column-to-text transformation
text X
pre-trained language model
vector X
shared weights
cosine(X, Y)
column Y
column-to-text transformation
text Y
pre-trained language model
vector Y
offline
searching
column X1
column X2
...
column-to-text transformation
text X1
text X2
...
pre-trained language model
vector X1
vector X2
...
ANN index
online
search
column Q
column-to-text transformation
text Q
pre-trained language model
vector Q

Figure 1: Overview of the DeepJoin model.

Table 1: Column-to-text tranformation options.

Name	Pattern
`col`	`$cell_1$`, `$cell_2$`, . . . , `$cell_n$`
`colname-col`	`$column_name$` : `$col$`.
`colname-col-context`	`$colname-col$` : `$col$`. `$table_context$`
`colname-col-stat`	`$column_name$` contains `$n$` values (`$max_len$`, `$min_len$`, `$avg_len$`): `$col$`
`title-colname-col`	`$table_title$`. `$colname-col$`
`title-colname-col-context`	`$title-colname-col$`. `$table_context$`
`title-colname-col-stat`	`$table_title$`. `$colname-col-stat$`.

for join. For this reason, we expect an *approximate* answer sublinear in $|\mathcal{X}|$, and by encoding the query column to a fixed length, it can be *independent* of $|Q|$ during the index lookup.

Apart from exact solutions, LSH Ensemble [58] is an approximate solution to equi-joinability. It partitions the repository and computes MinHash sketeches [6] with parameters tuned for each partition. Unlike JOSIE, it targets the thresholded problem (i.e., $\frac{|Q \cap X|}{|Q|} \geq t$), which requires a user-specified threshold and thus is less flexible than computing top-k. Although adaptation for the top-k problem is available, it suffers from the false positives introduced by transforming overlap similarity to Jaccard similarity for the use of MinHash, and runs even slower than JOSIE [56].

3 THE DEEPJOIN MODEL

Figure 1 illustrates the overview of the DeepJoin model. In DeepJoin, the joinable table discovery is essentially an embedding-based retrieval, which has recently been adopted in various retrieval tasks, such as long text retrieval [31] and search in social networks [25]. Embedding-based retrieval usually employs metric learning or meta embedding search to learn embeddings for target data such that the semantics can be measured in the embedding space, especially when target data are highly sparse or the semantics is hard to define by an analytical model. Another key benefit is, by embedding original data objects (i.e., columns) to a fixed-length vector, the subsequent search can be independent of the size of the data object, hence achieving our goal stated in Section 2.2.

Since semantic joins are also in our scope, an immediate idea is employing a PLM to capture the semantics. PLMs, such as BERT [12], have gained popularity in various data management tasks. By unsupervised training for language prediction tasks on large text corpora such as Wikipedia pages, these models are able to embed texts to a vector space such that texts are expected to be similar in meaning if they are close in the vector space. A salient property of PLMs is that they are deep models that can be tailored to downstream tasks (e.g., data preparation [46], entity matching [33], and column annotation [45]) by fine-tuning with task-specific training data. Moreover, state-of-the-art PLMs are mainly transfomer networks [47], which are deep neural networks that utilize the attention mechanism to focus on informative words than stop words. The attention mechanism can be also used in our problem, so that the model can focus on the cells that yield a match for joins, assuming that the query column has a similar cell distribution to those in the repository. As such, we are able to use one model design to cope with both equi-joins and semantic joins. The only difference is that the model is trained with a dataset labeled for the target join type. In DeepJoin, we use a PLM to embed columns to a vector space such that columns with high joinability are close in the vector space. Since PLMs take as input raw text, we transform the contents in each column to a text sequence (i.e., contextualize the column), and then feed the sequence to a PLM to produce a column embedding.

3.1 Column-to-Text Transformation

The column-to-text transformation belongs to prompt engineering, which works by including the description of the task in the input to train language models and has shown effectiveness in NLP tasks such as question answering [50]. In DeepJoin, we take advantage of metadata and consider seven options, shown in Table 1, where variables are quoted in dollar sign. n denotes the number of distinct cell values of the column. `cell_i` denotes the value of the i-th cell of the column, with duplicate values removed. `stat` denotes the statistics of the column, including the maximum, minimum, and average numbers of words in a cell. `table_context` denotes the accompanied context of the table (e.g., a brief description of the table). Note that some patterns are also used for creating other patterns. For example, `col` stands for the concatenation of all the cell values, with a comma as delimiter, and `colname-col` stands for the column name followed by a colon and the content of `col`.

3.2 Column Embedding

In DeepJoin, we fine-tune two state-of-the-art PLMs: DistilBERT [42], a faster and lighter variant of BERT, and MPNet [44], which leverages the dependency among predicted tokens through permuted language modeling. We use sentence-transformers [40] to output a sentence embedding for a piece of input text. Since DeepJoin is independent of the choice of PLMs, we expect its performance can be improved by more advanced PLMs.

Similar to many embedding-based retrieval, in order scale to large repositorities, we resort to approximate nearest neighbor (ANN) search. The embeddings of the columns in $\mathcal{X}$ are indexed offline. At the search time, we find the approximate k nearest neighbors (kNN) of the query column embedding under Euclidean distance. In particular, we resort to hierarchical navigable small world (HNSW) [34], which is among the most prevalent approaches to finding approximate answers to approximate kNN search [39]. Since the search time complexity of HNSW is claimed to be logarithmic in the number of indexed objects [34], the search can be done with a time complexity sublinear in the number of columns. Moreover, for billion-scale datasets, we may use inverted index over product quantization (IVFPQ) method [29], and construct HNSW over the coarse quantizer of IVFPQ. Such choice has become the common practice for billion-scale kNN search (e.g., using the Faiss library [17]).

3.3 Complexity Analysis

The search consists of two parts: query enconding and ANN search. In query enconding, we first transform the column to text, with a time complexity of $O(|Q|)$, and then we feed the text to the PLM, with a time complexity of $O(|M||Q|)$, where $|M|$ denotes the model size. In ANN search, due to the use of HNSW, the time complexity is $O(vl \log |\mathcal{X}|$, where v is the maximum out-degree in HNSW's index and l is the dimensionality of column embedding. Compared to JOSIE and PEXESO, which are linear in $|\mathcal{X}| \cdot |Q|$, we reduce the time complexity to logarithmic in $|\mathcal{X}|$ and it is independent of $|Q|$ in the ANN index lookup. Although the query encoding is still linear in $|Q|$, the column embedding can be accelerated by GPUs.

4 MODEL TRAINING

To fine-tune the PLM for joinability table discovery, we initialize the embedding model with the parameters of the PLM, and then train the embedding model with our training data and loss function.

4.1 Training Data

Given a repository $\mathcal{X}$, we collect column pairs in $\mathcal{X}$ with high joinability as positive training examples. This can be done by a self-join over the repository with a threshold t, i.e., finding column pairs (X, Y) such that $X \in \mathcal{X}$, $Y \in \mathcal{X}$, and $jn(X, Y) \geq t$. To this end, we invoke a set similarity join [7] for equi-joins and use PEXESO for semantic joins.

In Defintions 2.1 and 2.3, joinability is insensitive to the order of cells in a column, whereas PLMs are order-sensitive in its input. In order to make our model learn that joinability is order-insensitive, we consider a data augmentation technique by shuffling the cells in the column. In particular, we pick a percentage (called shuffle rate) of pairs (X, Y) in the aforementioned positive examples, generate a random permutation of the cells of X, denoted by X', and insert (X', Y) to the set of positive examples. As such, the training set contains both (X, Y) and (X', Y), hence to suggest the order-insensitive joinability. Given a shuffle rate of r, out of all the positive examples, $r/(1 + r)$ of them are obtained from cell shuffle.

To define negative training examples, we choose to use in-batch negatives, an easy and memory-efficient way that reuses the negative examples in the batch and has demonstrated effectiveness in text retrieval tasks [31]. Given a batch of positive training examples $\{ (X_i, Y_i) \}$ (note that X_i may be a shuffled column), we assume each $(X_i, Y_j), i \neq j$ as a negative pair. Despite a very small chance that (X_i, Y_j) are joinable, this can be regarded as noise in the training data and our model is robust against this case. In our experiments, it also shows better empirical results than other options such as removing matching cells from positive examples.

4.2 Loss Function

Given a batch of N training examples $\{ (X_i, Y_i) \}$ as defined above, we minimize the multiple negative ranking loss [23], which measures the negative log-likehood of softmax normalized scores:

$$\begin{aligned} L(\mathbf{X}, \mathbf{Y}) &= -\frac{1}{N}\sum_{i=1}^{N} \log P_{\text{approx}}(Y_i \mid X_i) \\ &= -\frac{1}{N}\sum_{i=1}^{N} \left[S(X_i, Y_i), - \log \sum_{j=1}^{N} \exp\left(S(X_i, Y_j)\right) \right]. \end{aligned}$$

The above loss function is one of the prevalent options [38] for fine-tuning sentence-transformers [40]. For the score function $S(\cdot, \cdot)$, we choose cosine similarity of the column embeddings, which shows the best empirical results. The subtlty here is that in online searching, Euclidean distance is used instead for retrieval. We believe the difference is because that the length of embeddings is also useful in identifying joinable results.

5 EXPERIMENTS

Due to the space limitation, we report the most important experimental results here. We provide the following experiments in the

Table 2: Dataset statistics.

Dataset	$\lvert\mathcal{X}\rvert$	max $\lvert X\rvert$	min $\lvert X\rvert$	avg $\lvert X\rvert$	# positive samples
Webtable-train	30K	5454	5	20.77	190K (equi-), 220K (semantic)
Wikitable-train	30K	1197	5	18.58	490K (equi-), 540K (semantic)
Webtable-test	1M	6031	5	20.25	N/A
Wikitable-test	1M	3454	5	18.71	N/A

extended version [16]: (1) accuracy and efficiency with varying column size, (2) ablation study for semantic joins, (3) accuracy with varying vector matching threshold τ for semantic joins, and (4) efficiency with varying k.

5.1 Experimental Settings

Datasets. The following datasets are used in the evaluation.

- **Webtable** is a dataset of the WDC Web Table Corpus [41]. We use the English relational Web tables 2015 and for each table we extract the key column defined in the metadata.
- **Wikitable** is a dataset of relational tables from Wikipedia [3]. For each table, we take the column which contains the largest number of distinct values in the table.

Both contain metadata for table title, column name, and context, and have been used in previous works [2, 15, 48, 52, 56, 58]. Columns that are too short (< 5 cells) are removed. For semantic joins, fastText [18] is used to embed cells, Euclidean distance is used for distance function d, and the threshold τ for vector matching is 0.9.

In order to show that our method is able to learn from a small subset of the corpus and **generalize** well to large datasets, we randomly sample two subsets, of 30K and 1M columns, respectively, from each corpus, dubbed “xxx-train” and “xxx-test”. From the training set, we randomly sample column pairs whose $jn \geq 0.7$ as initial positive samples, where jn is defined using Equation 2 for equi-joins or Equation 3 for semantic joins. We then apply the techniques in Section 4.1 for data augmentation and making negative examples. The testing subset is used as the target columns for search. To make queries and avoid data leak, we randomly sample 50 columns from the original corpus except those in $\mathcal{X}$. The dataset statistics is given in Table 2.

Methods. We compare the following methods.

- DeepJoin: This is our proposed model. We equi our model with DistilBERT [42] and MPNet [44] as PLM and denote the resultant model as $\text{DeepJoin}_{\text{DistilBERT}}$ and $\text{DeepJoin}_{\text{MPNet}}$, respectively.
- JOSIE [56]: This is an exact solution to equi-joinable table discovery, based on top-k set similarity search.
- LSH Ensemble [58]: This is an approximat solution to equi-joinable table discovery, based on partitioning and MinHash.
- fastText, BERT, MPNet: We replace the column embedding in DeepJoin by a simple average over the word embeddings output by fastText [18], BERT [12], and MPNet [44], respectively.
- TaBERT [52]: This is a table embedding approach which uses BERT and learns column embeddings for question answering tasks. We use its column embedding to replace that in DeepJoin.
- MLP: We replace the PLM in DeepJoin with a 3-layer perceptron trained for a regression, which takes as input the fastText embeddings of two columns and outputs the joinability. Then, we take the vector output by the last hidden layer as column embedding.
- PEXESO [15]: This is an exact solution to the semantic join defined in Section 2.1, based on a pivot index to search for top-k.

Metrics. For accuracy, we evaluate **precision@**k and normalized discounted cumulative gain (**NDCG@**k). Precision@k measures the overlap between the top-k results output by the model and the ground truth top-k output by an exact solution to Problem 1. NDCG@k is defined as $\frac{DCG_{\text{model}}}{DCG_{\text{ground\_truth}}}$, where $DCG = \sum_{i=1}^{k} \frac{jn(Q,X_i)}{\log_2(i+1)}$, and the X_i's for DCG_{model} and $DCG_{\text{ground\_truth}}$ are the top-k results of the model and the ground truth, respectively.

To evaluate the effectiveness of semantic joins, we also request our colleagues of database researchers to label whether a retrieved table is joinable, and measure precision, recall, and F1. Precision = (# retrieved joinable tables) / (# retrieved tables). Since it is too laborious to label every table in the dataset, we follow [28] and build a retrieved pool using the union of the tables identified by the competitors: Recall = (# retrieved joinable tables) / (# joinable tables in the retrieved pool).

For efficiency, we evaluate the end-to-end processing time, including column-to-text transformation, query embedding, and ANN search. Accuracy and efficiency are reported by averaging over all the queries used in the experiments.

Environments. DeepJoin are implemented in Pytorch, the Hugging Face Transformers library [1], and the Sentence-BERT library [40]. We use the following hyperparameter setting: batch size = 32, learning rate = 2e-5, warmup steps = 10000, and weight decay rate = 0.01. The methods involving column embedding (fastText, BERT, MPNet, TaBERT, MLP, and DeepJoin) follow the same ANN search scheme as DeepJoin. As described in Section 3.2, we use IVFPQ [29] plus HNSW [34], which are implemented in the Faiss library [17]. Experiments are run on a server with a 2.20GHz Intel Xeon CPU E7-8890 and 630 GB RAM. Training and inference of models are (optionally) accelerated using a single NVidia A100 Tensor Core. All the competitors are implemented in Python 3.7.

5.2 Accuracy Evaluation

For equi-join, Table 3 reports the precision and the NDCG for k from 10 to 50. JOSIE is omitted as it returns exact answers. For most competitors, the general trend is that both precision and NDCG increase with k. DeepJoin outperforms alternatives for all the settings. The best performance is observed when MPNet is equipped. LSH Ensemble's performance is mediocre due to the conversion from overlap condition to Jaccard condition, which becomes very loose when the lengths of query and target significantly differ. For embedding methods, fastText and PLMs (BERT and MPNet) are generally better than TaBERT. This is because TaBERT is trained using query answering rather than joinable table discovery. fastText is roughly better than BERT, and MPNet, indicating that simply using PLMs for joinable table discovery does not yield better accuracy than context-insensitive word embeddings. In addition, MLP's generally performs the best among the methods other than DeepJoin, showing that word embedding plus a regression delivers better results than using only language models.

Table 3: Accuracy of equi-joins.

Webtable										
LSH Ensemble	0.634	0.647	0.656	0.676	0.688	0.715	0.714	0.701	0.702	0.698
fastText	0.680	0.726	0.752	0.754	0.773	0.731	0.721	0.743	0.748	0.764
BERT	0.652	0.695	0.712	0.722	0.729	0.698	0.713	0.708	0.707	0.708
MPNet	0.610	0.629	0.644	0.649	0.654	0.674	0.677	0.678	0.680	0.677
TaBERT	0.622	0.637	0.645	0.656	0.671	0.694	0.685	0.690	0.693	0.691
MLP	0.683	0.719	0.755	0.758	0.778	0.737	0.735	0.748	0.755	0.769
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.702	0.741	0.775	0.793	0.805	0.744	0.752	0.758	0.761	0.788
DeepJoin$_{\text{MPNet}}$ (ours)	**0.732**	**0.775**	**0.791**	**0.812**	**0.832**	**0.768**	**0.786**	**0.799**	**0.803**	**0.822**
Wikitable										
LSH Ensemble	0.480	0.450	0.466	0.470	0.474	0.714	0.688	0.681	0.674	0.672
fastText	0.574	0.551	0.581	0.605	0.621	0.799	0.794	0.791	0.793	0.791
BERT	0.436	0.460	0.497	0.520	0.541	0.719	0.721	0.731	0.736	0.740
MPNet	0.442	0.464	0.504	0.524	0.543	0.711	0.721	0.729	0.735	0.736
TaBERT	0.431	0.445	0.488	0.520	0.539	0.701	0.708	0.732	0.725	0.737
MLP	0.578	0.576	0.585	0.610	0.619	0.801	0.802	0.800	0.804	0.802
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.588	0.593	0.612	0.635	0.807	0.813	0.822	0.825	0.823	0.827
DeepJoin$_{\text{MPNet}}$ (ours)	**0.614**	**0.622**	**0.641**	**0.666**	**0.678**	**0.821**	**0.824**	**0.830**	**0.833**	**0.833**

Table 4: Accuracy of semantic joins (labeled by PEXESO [15]).

Webtable										
	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
LSH Ensemble	0.696	0.670	0.613	0.554	0.508	0.578	0.599	0.615	0.618	0.626
fastText	0.842	0.917	0.945	0.957	0.964	0.575	0.588	0.631	0.647	0.647
DeepJoin (ours)	**0.874**	**0.934**	**0.954**	**0.963**	**0.970**	**0.640**	**0.657**	**0.664**	**0.685**	**0.680**
Wikitable										
LSH Ensemble	0.578	0.611	0.581	0.570	0.567	0.633	0.655	0.660	0.669	0.678
fastText	0.543	0.610	0.645	0.669	0.721	0.353	0.353	0.358	0.370	0.371
DeepJoin (ours)	**0.813**	**0.881**	**0.889**	**0.889**	**0.936**	**0.814**	**0.820**	**0.833**	**0.842**	**0.852**

Table 5: Accuracy of semantic joins (labeled by experts).

Webtable			
Methods	Precision	Recall	F1
LSH Ensemble	0.181	0.228	0.202
fastText	0.138	0.277	0.183
PEXESO	0.212	0.506	0.300
DeepJoin (ours)	**0.350**	**0.693**	**0.465**
Wikitable			
LSH Ensemble	0.652	0.385	0.484
fastText	0.467	0.380	0.419
PEXESO	0.683	0.492	0.572
DeepJoin (ours)	**0.842**	**0.568**	**0.677**

For semantic join, Table 4 reports the precision and NDCG evaluated under PEXESO's definition (Definition 2.3). Since MPNet always reports better results than DistilBERT, in the rest of the experiments, we only show the results equipped with MPNet for DeepJoin. Similar to equi-join, DeepJoin delivers better results than alternatives for all the settings. fastText is the runner-up on Webtable but is not good on Wikitable. We then evaluate these methods using the labels from our DB experts. The precision, recall, and F1 measure are reported in Table 5. DeepJoin still performs the best. It is even better than PEXESO, and the advantage is remarkable (by a margin of 0.105 − 0.165 in F1). We believe there are two reasons. First, DeepJoin uses a fine-tuned PLM, which captures the semantics of table contents in a better way than PEXESO, which uses fastText to embed cell values. Second, PEXESO defines matching cells using a threshold. When judged by experts for joinability, the matching condition may differ across cell values, queries, and target columns, while a fixed threshold may not fit all of them.

5.3 Ablation Study

For ablation study, we report the results for equi-joins here and provide the results for semantic joins in the extended version [16].

We first evaluate the impact of column-to-text transformation and test the seven options proposed in Section 3.1. The results are reported in Table 6. Adding column name at the beginning (`colname-col`) improves the performance of simply concatenating cell values (`col`). Similarly, adding table title at the beginning (options with `title`) also has a positive impact. Appending statistical information (options with `stat`) further improves the performance, whereas appending context (options with `context`) has a negative impact. The latter is because the context includes information irrelevant to the column. Among the seven options, `title-colname-col-stat` is best.

Table 6: Evaluation of column-to-text transformation, equi-join.

Webtable										
	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
`col`	0.700	0.744	0.763	0.788	0.791	0.745	0.753	0.767	0.779	0.795
`colname-col`	0.709	0.750	0.771	0.795	0.799	0.751	0.757	0.770	0.785	0.802
`colname-col-context`	0.703	0.746	0.764	0.795	0.798	0.750	0.755	0.770	0.780	0.800
`colname-col-stat`	0.712	0.757	0.778	0.799	0.799	0.756	0.758	0.773	0.788	0.805
`title-colname-col`	0.729	0.771	0.785	0.807	0.821	0.761	0.769	0.788	0.795	0.818
`title-colname-col-context`	0.718	0.759	0.781	0.799	0.820	0.759	0.766	0.784	0.791	0.815
`title-colname-col-stat`	**0.732**	**0.775**	**0.791**	**0.812**	**0.832**	**0.768**	**0.786**	**0.799**	**0.803**	**0.822**
Wikitable										
`col`	0.602	0.604	0.617	0.632	0.651	0.804	0.805	0.812	0.819	0.821
`colname-col`	0.600	0.607	0.615	0.630	0.654	0.801	0.816	0.817	0.821	0.822
`colname-col-context`	0.599	0.607	0.613	0.628	0.655	0.805	0.814	0.818	0.819	0.821
`colname-col-stat`	0.605	0.608	0.617	0.635	0.663	0.801	0.814	0.815	0.822	0.824
`title-colname-col`	0.611	0.614	0.627	0.647	0.671	0.813	0.820	0.824	0.827	**0.833**
`title-colname-col-context`	0.608	0.618	0.630	0.644	0.670	0.815	0.821	0.822	0.828	0.831
`title-colname-col-stat`	**0.614**	**0.622**	**0.641**	**0.666**	**0.678**	**0.821**	**0.824**	**0.830**	**0.833**	**0.833**

Table 7: Evaluation of cell shuffle, equi-join.

Webtable										
	Precision@k					NDCG@k				
shuffle rate	k = 10	20	30	40	50	k = 10	20	30	40	50
`no-shuffle`	0.720	0.759	0.781	0.803	0.819	0.752	0.771	0.784	0.791	0.812
`0.1`	0.725	0.766	0.784	0.809	0.825	0.755	0.778	0.793	0.796	0.817
`0.2`	**0.732**	**0.775**	**0.791**	**0.812**	**0.832**	**0.768**	**0.786**	**0.799**	**0.803**	**0.822**
`0.3`	0.729	0.770	0.785	0.792	0.815	0.754	0.773	0.788	0.791	0.806
`0.4`	0.711	0.755	0.774	0.780	0.782	0.733	0.758	0.766	0.780	0.781
`0.5`	0.701	0.751	0.760	0.781	0.787	0.726	0.754	0.760	0.765	0.777
Wikitable										
`no-shuffle`	0.605	0.615	0.631	0.657	0.670	0.811	0.813	0.815	0.826	0.821
`0.1`	0.608	0.618	0.635	659	0.675	0.809	0.814	0.829	0.828	0.829
`0.2`	0.611	**0.622**	664	0.639	0.677	0.815	0.820	0.831	0.832	0.830
`0.3`	**0.614**	**0.622**	**0.641**	**0.666**	**0.678**	**0.821**	**0.824**	**0.830**	**0.833**	**0.833**
`0.4`	0.584	0.598	0.613	0.634	0.644	0.803	0.801	0.813	0.815	0.821
`0.5`	0.576	0.579	0.591	0.623	0.634	0.800	0.797	0.802	0.808	0.810

Table 8: Processing time per query, varying dataset size.

	query encoding (ms)	total (ms)				
Methods		$|\mathcal{X}|$ = 1M	2M	3M	4M	5M
Webtable, equi-join						
LSH Ensemble	-	508	597	634	689	785
JOSIE	-	506	751	874	980	1103
fastText	9	9.7	10.3	11.5	12.1	13.6
DeepJoin (CPU)	66	68.1	69.3	71.4	73.2	74.1
DeepJoin (GPU)	**7**	**8.0**	**8.7**	**9.6**	**10.8**	**10.7**
Webtable, semantic join						
PEXESO	-	2566	3116	3780	4122	4590
DeepJoin (CPU)	74	76.1	77.9	78.4	80.1	79.9
DeepJoin (GPU)	**7**	**8.4**	**8.8**	**9.5**	**9.7**	**10.9**

We then evaluate the impact of cell shuffle for data augmentation. We vary the shuffle rate (defined in Section 4.1) and report the accuracy in Table 7. We observe that a moderate shuffle rate achieves the best performance (0.2 for Webtable and 0.3 for Wikitable), indicating that shuffling the cells in columns helps the model learn that joinability is order-insensitive. On the other hand, over-shuffling is negative and even worse than no shuffle. We suspect this is because the original order of cells in both datasets follows some distribution. The attention mechanism in the PLM is able to capture such distribution and focus on the cells that are more probable to match. When the order is too random, the attention mechanism loses focus and thus a detrimental impact is observed.

5.4 Efficiency Evaluation

We vary the repository size from 1M to 5M and report the average query processing time on Webtables in Table 8. For embedding methods, we also report query encoding time, which includes column-to-text transformation and column embedding. JOSIE and PEXESO are the slowest for equi-joins and semantic joins, respectively, and both exhibit substantial growth of search time (around 2X) when we increase the dataset size from 1M to 5M. LSH Ensemble is also slow, despite transforming columns to fixed-length sketches. In contrast, embedding methods are much faster, though the majority of time is spent on query encoding. For example, DeepJoin, even if only equipped with CPU, is 7 – 57 times faster than the above methods. The growth of search time is also slight (1.09 times for equi-joins and 1.05 times for semantic joins, when $|\mathcal{X}|$ increases from 1M to 5M), showing the scalability of embedding methods. With the help of GPU, DeepJoin can be substantially accelerated and become two-orders of magnitude faster than exact solutions JOSIE and PEXESO, and even faster than fastText.

6 RELATED WORK

Table discovery and data lake management. Besides joinable table discovery [56, 58], techniques have been developed for searching unionable tables [37]. Another important problem is related table discovery. SilkMoth [10] models columns as sets and finds related sets under maximum bipartite matching metrics. JUNEAU [55] finds related tables for data science notebooks using a composite score of multiple similarities. D^3L [5] is a dataset discovery method which finds top-k results with a scoring function involving multiple attributes of a table. DLN [4] discovers related datasets by exploiting historical queries having join clauses and determines relatedness with a random forest. Another notable approach is Nextia$_{JD}$ [19], which uses meta-features (cardinalities, value distribution, entropy, etc.) to provide a join quality ranking of candidate columns.

For data lake management, another problem which has been extensively studied is column type annotation. Recent approaches include Sherlock [26], Sato [53], and DODUO [45]. Among the three, DODUO is the one that employs PLMs. To deal with the case when tables differ in format, transformation techniques are often used to convert data so they can be joined. To tackle this problem, auto-join [57] joins two tables with string transformations on columns. A similar method is auto-transform [22], which learns string transformations with patterns. Moreoever, SEMA-join [21] finds related pairs between two tables with statistical correlation. Other representative problems of data lake managment include data lake organization [36] and data validation [43].

Table embedding. Recently, language models have been used in understanding the contents of tables, and various table representation learning methods have been developed. For example, the cell classification problem was investigated in [20], with an RNN-based cell embedding method proposed. Table2Vec [54], which features a series of embedding approaches for words, entities, and headers based on the idea of skip-gram model of word2vec [35], deals with the table retrieval problem that returns a ranked list of tables for a keyword query. In addition, PLMs such as BERT [12] has also been used for table retrieval [9], accompanied by an embedding-based feature selection technique to select relevant contents to a query and an MLP for relevance score computation.

More advanced approaches enlarged the scope of downstream tasks to include entity linkage, column type annotation, cell filling, etc., and designed pre-trained models that can be fine-tuned for these tasks. TURL [11] features a contextualization technique to convert table contents to sequences and leverages a masked language model (MLM), which is initialized by TinyBERT [30]. TaPas [24] is based on a similar MLM but employs BERT [12], with additional information embedded such as positions and ranks. TaBERT [52] builds its model upon question answering tasks and learns embeddings for cells, columns, and utterance tokens in the questions using BERT. By adopting two transformers [47] to independently encode rows and columns, TABBIE [27] is able to embed cells, columns, and rows, and achieves one order of magnitude less training time than TaBERT. As a tree-based model based on BERT, TUTA [49] creates trees to encode the hierarchical information in hierarchical tables, while most previous studies focused on flat tables. Another method for hierarchical tables is GTR [48], which models cells, rows, and columns as nodes in graphs and employs a graph transformer [32] to catpure the neighborhood information of cells and the global information of rows and columns.

7 CONCLUTION

In this paper, we proposed DeepJoin, a deep learning model for joinable table discovery. DeepJoin is able to handle both equi- and semantic joins with the same model design. We design the model in an embedding-based retrieval fashion, which embeds columns to vectors with a PLM and resorts to approximate nearest neighbor search to find joinable results, hence achieving a search time sublinear in the repository size. A metric learning approach is proposed so that columns are expected to be joinable if they are close in the embedding space. We also devise techniques for contextualization and training data preparation. The experiments show the effectiveness and the generalizability of DeepJoin, which is consistently more accurate than alternative methods and even better than the exact solution to semantic join when evaluated on labels from DB experts. The evaluation also shows the superiority of DeepJoin in search speed, which can be up to two-order-of magnitude faster than exact solutions. We expect that by employing more advanced retrieval strategies such as a two-stage retrieval, the accuracy of DeepJoin can be further improved.

REFERENCES

[1] 2022. Hugging Face Transformers. https://huggingface.co/docs/transformers/main/en/index.

[2] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *ISWC (Lecture Notes in Computer Science)*, Vol. 9366. Springer, 425–441. https://doi.org/10.1007/978-3-319-25007-6_25

[3] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. Wikitables. http://websail-fe.cs.northwestern.edu/TabEL/.

[4] Sagar Bharadwaj, Praveen Gupta, Ranjita Bhagwan, and Saikat Guha. 2021. Discovering Related Data At Scale. *PVLDB* 14, 8 (2021), 1392–1400. https://doi.org/10.14778/3457390.3457403

[5] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *ICDE*. 709–720. https://doi.org/10.1109/ICDE48307.2020.00067

[6] Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *SEQUENCES*, Bruno Carpentieri, Alfredo De Santis, Ugo Vaccaro, and James A. Storer (Eds.). IEEE, 21–29. https://doi.org/10.1109/SEQUEN.1997.666900

[7] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. 2006. A Primitive Operator for Similarity Joins in Data Cleaning. In *ICDE*, Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang (Eds.). IEEE Computer Society, 5. https://doi.org/10.1109/ICDE.2006.9

[8] Lu Chen, Yunjun Gao, Baihua Zheng, Christian S. Jensen, Hanyu Yang, and Keyu Yang. 2017. Pivot-based Metric Indexing. *PVLDB* 10, 10 (2017), 1058–1069. https://doi.org/10.14778/3115404.3115411

[9] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D. Davison. 2020. Table Search Using a Deep Contextualized Language Model. In *SIGIR*, Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (Eds.). ACM, 589–598. https://doi.org/10.1145/3397271.3401044

[10] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. 2017. SilkMoth: An Efficient Method for Finding Related Sets with Maximum Matching Constraints. *PVLDB* 10, 10 (2017), 1082–1093. https://doi.org/10.14778/3115404.3115413

[11] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *PVLDB* 14, 3 (2020), 307–319. https://doi.org/10.5555/3430915.3442430

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[13] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration.* Morgan Kaufmann. http://research.cs.wisc.edu/dibook/

[14] Yuyang Dong and Masafumi Oyamada. 2022. Table Enrichment System for Machine Learning. In *SIGIR*, Enrique Amigó, Pablo Castells, Julio Gonzalo, Ben Carterette, J. Shane Culpepper, and Gabriella Kazai (Eds.). ACM, 3267–3271. https://doi.org/10.1145/3477495.3531678

[15] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. In *ICDE*. IEEE, 456–467. https://doi.org/10.1109/ICDE51399.2021.00046

[16] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2022. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. https://dongyuyang.github.io/papers/deep-join-full.pdf/.

[17] facebook AI research. 2022. Faiss: Facebook AI Similarity Search. https://github.com/facebookresearch/faiss/wiki/Faiss-indexes.

[18] Facebook AI Research Lab. 2015. fastText: Library for Efficient Text Classification and Representation Learning. https://fasttext.cc/.

[19] Javier Flores, Sergi Nadal, and Oscar Romero. 2021. Effective and Scalable Data Discovery with NextiaJD. In *EDBT*, Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra (Eds.). OpenProceedings.org, 690–693. https://doi.org/10.5441/002/edbt.2021.85

[20] Majid Ghasemi-Gol, Jay Pujara, and Pedro A. Szekely. 2019. Tabular Cell Classification Using Pre-Trained Cell Embeddings. In *ICDM*, Jianyong Wang, Kyuseok Shim, and Xindong Wu (Eds.). IEEE, 230–239. https://doi.org/10.1109/ICDM.2019.00033

[21] Yeye He, Kris Ganjam, and Xu Chu. 2015. SEMA-JOIN: Joining Semantically-Related Tables Using Big Table Corpora. *PVLDB* 8, 12 (2015), 1358–1369. https://doi.org/10.14778/2824032.2824036

[22] Yeye He, Zhongjun Jin, and Surajit Chaudhuri. 2020. Auto-Transform: Learning-to-Transform by Patterns. *PVLDB* 13, 11 (2020), 2368–2381. http://www.vldb.org/pvldb/vol13/p2368-he.pdf

[23] Matthew L. Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient Natural Language Response Suggestion for Smart Reply. *CoRR* abs/1705.00652 (2017). arXiv:1705.00652 http://arxiv.org/abs/1705.00652

[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*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 4320–4333. https://doi.org/10.18653/v1/2020.acl-main.398

[25] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based Retrieval in Facebook Search. In *KDD*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 2553–2561. https://doi.org/10.1145/3394486.3403305

[26] Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *KDD*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 1500–1508. https://doi.org/10.1145/3292500.3330993

[27] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *NAACL-HLT*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 3446–3456. https://doi.org/10.18653/v1/2021.naacl-main.270

[28] Clarke Sarah J. and Willett Peter. 1997. Estimating the recall performance of Web search engines. *Aslib Proceedings* 49, 7 (01 Jan 1997), 184–189. https://doi.org/10.1108/eb051463

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

[30] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. In *EMNLP (Findings) (Findings of ACL)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.), Vol. EMNLP 2020. Association for Computational Linguistics, 4163–4174. https://doi.org/10.18653/v1/2020.findings-emnlp.372

[31] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6769–6781. https://doi.org/10.18653/v1/2020.emnlp-main.550

[32] Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. 2019. Text Generation from Knowledge Graphs with Graph Transformers. In *NAACL-HLT*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 2284–2293. https://doi.org/10.18653/v1/n19-1238

[33] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* 14, 1 (2020), 50–60. https://doi.org/10.14778/3421424.3421431

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

[35] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (Eds.). 3111–3119. https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[36] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. 2020. Organizing Data Lakes for Navigation. In *SIGMOD*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1939–1950. https://doi.org/10.1145/3318464.3380605

[37] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *PVLDB* 11, 7 (2018), 813–825. https://doi.org/10.14778/3192965.3192973

[38] Nils Reimers. 2019. SentenceTransformers.Losses. https://www.sbert.net/docs/package_reference/losses.html.

[39] Jianbin Qin, Wei Wang, Chuan Xiao, Ying Zhang, and Yaoshu Wang. 2021. High-Dimensional Similarity Query Processing for Data Science. In *KDD*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 4062–4063. https://doi.org/10.1145/3447548.3470811

[40] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 3980–3990. https://doi.org/10.18653/v1/D19-1410

[41] Dominique Ritze, Oliver Lehmberg, Robert Meusel, Christian Bizer, and Sanikumar Zope. 2015. WDC Web Table Corpus. http://webdatacommons.org/webtables/2015/downloadInstructions.html.

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

[43] Jie Song and Yeye He. 2021. Auto-Validate: Unsupervised Data Validation Using Data-Domain Patterns Inferred from Data Lakes. In *SIGMOD*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 1678–1691. https://doi.org/10.1145/3448016.3457250

[44] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. MPNet: Masked and Permuted Pre-training for Language Understanding. In *NeurIPS*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan,

and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/c3a690be93aa602ee2dc0ccab5b7b67e-Abstract.html

[45] 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 *SIGMOD*, Zachary Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 1493–1503. https://doi.org/10.1145/3514221.3517906

[46] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *PVLDB* 14, 8 (2021), 1254–1261. https://doi.org/10.14778/3457390.3457391

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[48] Fei Wang, Kexuan Sun, Muhao Chen, Jay Pujara, and Pedro A. Szekely. 2021. Retrieving Complex Tables with Multi-Granular Graph Representation Learning. In *SIGIR*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 1472–1482. https://doi.org/10.1145/3404835.3462909

[49] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. TUTA: Tree-based Transformers for Generally Structured Table Pre-training. In *KDD*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 1780–1790. https://doi.org/10.1145/3447548.3467434

[50] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *CoRR* abs/2201.11903 (2022). arXiv:2201.11903 https://arxiv.org/abs/2201.11903

[51] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. 2011. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.* 36, 3 (2011), 15:1–15:41. https://doi.org/10.1145/2000824.2000825

[52] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 8413–8426. https://doi.org/10.18653/v1/2020.acl-main.745

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

[54] Li Zhang, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *SIGIR*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 1029–1032. https://doi.org/10.1145/3331184.3331333

[55] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *SIGMOD*. 1951–1966. https://doi.org/10.1145/3318464.3389726

[56] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 847–864. https://doi.org/10.1145/3299869.3300065

[57] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-Join: Joining Tables by Leveraging Transformations. *PVLDB* 10, 10 (2017), 1034–1045. https://doi.org/10.14778/3115404.3115409

[58] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *PVLDB* 9, 12 (2016), 1185–1196. https://doi.org/10.14778/2994509.2994534