DeepJoin: Joinable Table Discovery with Pre-trained Language Models

Yuyang Dong
NEC Corporation
dongyuyang@nec.com

Chuan Xiao
Osaka University
Nagoya University
chuanx@ist.osaka-u.ac.jp

Takuma Nozawa
NEC Corporation
nozawa-takuma@nec.com

Masafumi Enomoto
NEC Corporation
masafumi-enomoto@nec.com

Masafumi Oyamada
NEC Corporation
oyamada@nec.com

ABSTRACT

Due to the usefulness in data enrichment for data analysis tasks, joinable table discovery has become an important operation in data lake management. Existing approaches target equi-joins, the most common way of combining tables for creating a unified view, or semantic joins, which tolerate misspellings and different formats to deliver more join results. They are either exact solutions whose running time is linear in the sizes of query column and target table repository, or approximate solutions lacking precision. In this paper, we propose DeepJoin, a deep learning model for accurate and efficient joinable table discovery. Our solution is an embedding-based retrieval, which employs a pre-trained language model (PLM) and is designed as one framework serving both equi- and semantic (with a similarity condition on word embeddings) joins for textual attributes with fairly small cardinalities. We propose a set of contextualization options to transform column contents to a text sequence. The PLM reads the sequence and is fine-tuned to embed columns to vectors such that columns are expected to be joinable if they are close to each other in the vector space. Since the output of the PLM is fixed in length, the subsequent search procedure becomes independent of the column size. With a state-of-the-art approximate nearest neighbor search algorithm, the search time is sublinear in the repository size. To train the model, we devise the techniques for preparing training data as well as data augmentation. The experiments on real datasets demonstrate that by training on a small subset of a corpus, DeepJoin generalizes to large datasets and its precision consistently outperforms other approximate solutions'. DeepJoin is even more accurate than an exact solution to semantic joins when evaluated with labels from experts. Moreover, when equipped with a GPU, DeepJoin is up to two orders of magnitude faster than existing solutions.

PVLDB Reference Format:
Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. PVLDB, 16(10): 2458 - 2470, 2023.
doi:10.14778/3603581.3603587

Proceedings of the VLDB Endowment, Vol. 16, No. 10 ISSN 2150-8097.
doi:10.14778/3603581.3603587

1 INTRODUCTION

Given a table repository and a query table with a specified join column, joinable table discovery finds the target tables that can be joined with the query. Due to the demonstrated usefulness in data enrichment [10, 16], joinable table discovery has become a key procedure in data lake management and serves various downstream applications, especially those involving data analysis.

For joinable table discovery, early attempts mainly targeted equi-joins [64, 66], which are the most common way of combining tables for creating a unified view [15] and can be easily implemented using SQL. To deliver more joins results for heterogeneous data, recent approaches [16, 17] studied semantic joins, which join on cells with similar meanings via word embedding, so as to handle data with misspellings and discrepancy in formats/terminologies (e.g., "American Indian & Alaska Native" v.s. "Mainland Indigenous"). There are two major limitations in these solutions. First, they only apply to a single join type. Second, most of them are exact algorithms with a worst-case time complexity linear in the product of query column size and table repository size, and thus their scalability is dubious. Despite the existence of an approximate algorithm for equi-joins [66], it is based on MinHash sketches [6] and has to convert the joinability condition to a Jaccard similarity condition, which is non-equivalent and introduces many false positives. Moreover, it is sometimes even slower than an exact algorithm [64].

Seeing the limitations of existing solutions, we propose DeepJoin, a deep learning model designed in a *two-birds-with-one-stone* fashion such that both equi- and semantic joins can be served with one framework. In particular, DeepJoin targets textual attributes with fairly small cardinalities that can fit with language models. To cope with semantic joins, it works on a similarity condition of word embeddings and finds similar textual columns as exactly as possible. To resolve the efficiency issue, it finds joinable tables via an **embedding-based retrieval**. In particular, we employ an embedding model to transform columns to a vector space. By metric learning, columns with high joinability are close to each other in the vector space. Then, to find the top-k target columns ranked by joinability, we resort to a state-of-the-art approximate nearest neighbor search (ANNS) algorithm [38], whose time complexity is *sublinear* in the table repository size.

DeepJoin utilizes a **pre-trained language model** (PLM) for column embedding. PLMs, such as BERT [14], have gained popularity in various data management tasks that involve natural language processing. A salient property of modern PLMs is that they are

transformer networks [55] featuring the attention mechanism, thus not only good at capturing the semantics of column contents for semantic joins, but also able to focus on the cells that are more probable to match in equi-joins, assuming that the query column has a similar distribution to those in the repository. As such, our model gains the capability of handling both join types, only needing the PLM to be fine-tuned on the data labeled for either equi- or semantic joins. In addition, PLMs produce a fixed-length vector, meaning that the following index lookup and search are *independent* of the column size, hence along with the ANNS, addressing the scalability issue in existing solutions. Since PLMs take a text sequence as input, by prompt engineering, we propose a set of options that contextualizes a column to a text sequence. To train the model in a self-supervised manner, we devise a series of techniques to effectively generate positive and negative examples. Moreover, our training features a data augmentation technique, through which our model can learn that the joinability is insensitive to the order of cells in a column.

We conduct experiments on two real datasets and evaluate DeepJoin equipped with two state-of-the-art PLMs. We show that by training on a small subset (30k columns) of the corpus, DeepJoin *generalizes* well to large datasets (1M columns). In particular, DeepJoin outperforms alternative approximate solutions in all the settings, and reports an average precision of 72% for equi-joins and 91% for semantic joins and an average NDCG of 81% for equi-joins and 75% for semantic joins. To test the effectiveness of semantic joins, we also evaluate DeepJoin using data labeled by our database researchers, and the results show that DeepJoin is even better by a margin of 0.105 – 0.165 F1 score than PEXESO [17], an exact solution we use to label DeepJoin's training data. An ablation study demonstrates the usefulness of the proposed contextualization and data augmentation techniques. For scalability test, we vary dataset size from 1M to 5M columns. Even if equipped with a CPU, DeepJoin exhibits superb scalability and is 7 – 57 times faster than existing solutions. With the help of a GPU, DeepJoin outperforms them by up to two orders of magnitude.

Contributions. (1) We propose DeepJoin, a framework for joinable search discovery in a data lake. Our solution targets targets textual attributes with fairly small cardinalities, and is able to detect both equi-joinable and semantically joinable tables. (2) We design the search in DeepJoin as an embedding-based retrieval which employs a fine-tuned PLM for column embedding and ANNS for fast retrieval. The search time complexity is sublinear in the table repository size, and except the column embedding, the search time is independent of the column size. (3) We propose a prompt engineering method to transform column contents to a text sequence fed to the PLM. (4) We devise techniques for training data preparation and data augmentation, as well as a metric learning method to fine-tune the PLM in a self-supervised manner. (5) We conduct experiments to show that our model generalizes well to large datasets and it is accurate and scalable in finding joinable tables.

Furthermore, we would like to mention the following facts: (1) For the embedding-based retrieval in our model, the results of ANNS are directly output as the results of joinable table discovery, whereas more advanced paradigms exist, such as two-stage retrieval [11] which finds a set of candidates by ANNS and ranks the candidates by a more sophisticated model. (2) DeepJoin is not limited to the two PLMs evaluated in our experiments, because the PLM can be regarded as a plug-in in our framework. As such, we expect the performance of DeepJoin can be further improved by using more advanced retrieval paradigms or PLMs.

2 PRELIMINARIES

2.1 Problem Definition

Given a data lake of tables, we extract all the columns in these tables, except those unlikely to 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. Next, we define the joinability for these two types, respectively.

Given a query column Q and a 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 at least one 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 cells in Q that match those in X, and thus can be used to measure the equi-joinability [64]. 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 and 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 [19]). 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 $M_\tau^d(v_1, v_2)$ to denote if v_1 matches v_2; i.e., $M_\tau^d(v_1, v_2) = 1$, 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 [23], 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,

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

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 [64] and PEXESO [17]) 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 discovery 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 repository of target columns $\mathcal{X}$, the joinable table discovery problem is to find the top-k columns in $\mathcal{X}$ with the highest joinability from Q. Formally, we find a subset $\mathcal{R} \subseteq \mathcal{X}$, $|\mathcal{R}| = k$, and* $\min\{\, jn(Q,X) \mid X \in \mathcal{R} \,\} \geq jn(Q,Y), \forall Y \in \mathcal{X} \backslash \mathcal{R}$.

In this paper, we focus on dealing with textual columns. For numerical columns, a typical solution is utilizing the statistical feature vector in Sherlock [28], which converts a numerical column to a vector, and then we can invoke a vector search to look for joinable columns having similar statistics to the query column.

2.2 State-of-the-Art

JOSIE [64], a 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 [59], which have been extensively used for solving set similarity queries. JOSIE creates an inverted index over $\mathcal{X}$, 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 retrieving 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 alternate manner, JOSIE features the techniques to determine the their order, so as to optimize for long columns and large token universes, which are often observed in data lakes.

PEXESO [17] 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 can be 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 limitation 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 repository, it relies on a user-specific threshold for the count of matching vectors. This does not apply to the top-k case, and the algorithm is downgraded to be linear in $|\mathcal{X}_V| \cdot |Q|$, because at the early stage of search, due to the low count of matching vectors in the temporary top-k results, the pruning power of the grid index is next to none.

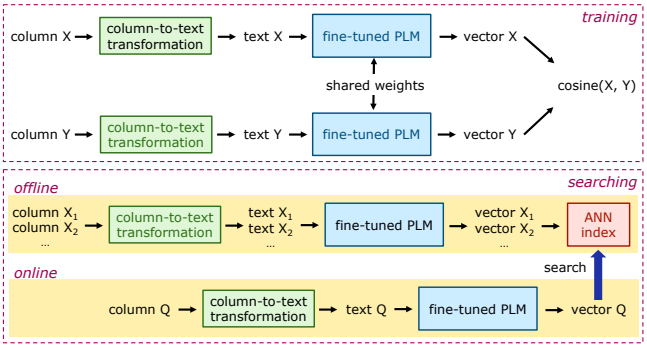

Figure 1: Overview of the DeepJoin model.

Table 1: Column-to-text transformation options.

Name	Pattern
col	$cell_1$, $cell_2$, . . ., $cell_n$
colname-col	$column_name$: col.
colname-col-context	$colname-col$: col. $table_context$
colname-stat-col	$column_name$ 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-stat-col	$table_title$. $colname-stat-col$.

In general, for search time, both JOSIE and PEXESO are linear in the product of column size and repository 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 data scientists are usually concerned with only part of the top-k results and will choose a subset of them for the join. For this reason, we will design our solution with the following two goals: (A) it returns an approximate answer with *sublinear* time in $|\mathcal{X}|$, and (B) by encoding the query column to a fixed length, it is *independent* of $|Q|$ and $\overline{|X|}$ during index lookup and search.

Apart from exact solutions, LSH Ensemble [66] is an approximate solution to equi-joinability. It partitions the repository and computes MinHash sketches [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 t and thus is less flexible than computing top-k. Although adaptation for the top-k problem is available, it suffers from low precision due to the many false positives introduced by transforming overlap similarity to Jaccard similarity for the use of MinHash, and it sometimes runs even slower than JOSIE [64].

3 THE DEEPJOIN MODEL

Figure 1 illustrates the overview of our 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 [33] and search in social networks [27]. An 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

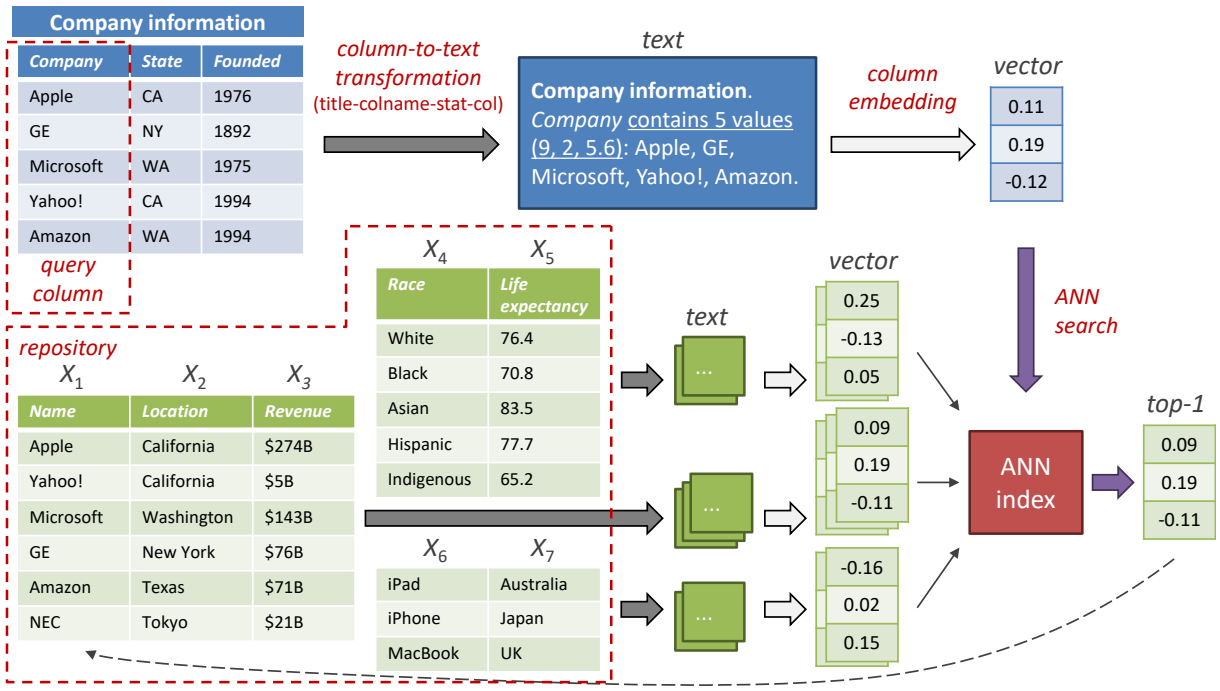

Figure 2: A running example of DeepJoin over a repository of 7 columns, $k = 1$.

embedding original data objects (i.e., columns) to a fixed-length vector, the subsequent procedure can be independent of the size of the data object, thereby achieving goal (B) stated in Section 2.2.

Since semantic joins are also in our scope, an immediate idea is employing a PLM to capture the semantics. By unsupervised training for language prediction tasks on a large text corpora such as Wikipedia pages, PLMs are able to embed texts to a vector space such that texts are expected to be similar in meaning if they are closer in the vector space. A key advantage of PLMs is that they are deep models that can be tailored to downstream tasks (e.g., data preparation [54], entity matching [37], and column annotation [52]) by fine-tuning with task-specific training data. Moreover, state-of-the-art PLMs utilize the attention mechanism to focus on informative words than stop words. Besides handling the semantics, the attention mechanism can be also useful for identifying equi-joinable columns, because it can focus on the cells that are more probable to yield a match for equi-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 framework to cope with both equi-joins and semantic joins. The only difference is that the model is trained with data labeled for the target join type. In DeepJoin, we use a fine-tuned PLM to embed columns to a vector space such that columns with high joinability are close to each other in the vector space. Since PLMs take as input raw text, we transform (i.e., contextualize) the contents in each column to a text sequence, and then feed the sequence to the fine-tuned 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 a model and has shown effectiveness in natural language processing tasks such as question answering [58]. In DeepJoin, we take advantage of metadata and consider seven options, shown in Table 1, where variables are quoted in dollar signs. In particular, `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; and `table_context` denotes the accompanied context of the table (e.g., a brief description). 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`.

Example 3.1. Consider the query column in Figure 2. Suppose `title-colname-stat-col` is used for column-to-text transformation. The table title is "Company information". The column name is "Company". There are 5 cell values: "Apple", "GE", "Microsoft", "Yahoo!", and "Amazon", with a maximum of 9 characters, a minimum of 2 characters, and an average of 5.6 characters. Therefore, according to the patterns shown in Table 1, the column is transformed to a text sequence "Company information. Company contains 5 values (9, 2, 5.6): Apple, GE, Microsoft, Yahoo!, Amazon.", as shown in Figure 2. For the repository, we can transform the 7 target columns to 7 text sequences using the same technique.

3.2 Column Embedding

In DeepJoin, we fine-tune two state-of-the-art PLMs: DistilBERT [49], a faster and lighter variant of BERT [14], and MPNet [51], which leverages the dependency among predicted tokens through permuted language modeling and takes auxiliary position information

as input to make the model see a full sentence, thereby reducing position discrepancy. We use sentence-transformers [47] to output a sentence embedding for a sequence of input text. It is also noteworthy to mention that for semantic joins, unlike PEXESO, we do not need to generate embeddings in the vector space $\mathcal{V}$ (Definition 2.2). Instead, the semantics is captured by the fine-tuned PLM.

Since PLMs have an input length limit max_seq_length (e.g., 512 tokens for BERT), in the case of a tall input column, we choose a frequency-based approach, e.g., taking a sample of the most frequent cell values from the column, whose number of tokens is just within max_seq_length. Then, we use the sample for column-to-text transformation. The reason is that they are more likely to yield join results. Here, the frequency of a cell value is defined as document frequency, i.e., the number of target columns in the repository that have this cell value. If columns are modeled as multisets, we may resort to other frequency-based approaches, such as TF-IDF and BM25 [39].

3.3 Indexing and Searching

In order to scale to large table repositories, we resort to approximate nearest neighbor search (ANNS). The embeddings of the columns in $\mathcal{X}$ are indexed offline. For online search, we find the k nearest neighbors (kNN) of the query column embedding under Euclidean distance as search results. In particular, we use hierarchical navigable small world (HNSW) [38], which is among the most prevalent approaches to ANNS [45]. Since the search time complexity of HNSW is sublinear in the number of indexed objects [38], the search can be done with a time complexity sublinear in the number of target columns, thereby achieving goal (A) stated in stated in Section 2.2. Moreover, for billion-scale datasets, we may use an inverted index over product quantization (IVFPQ) [31], and construct HNSW over the coarse quantizer of IVFPQ. Such choice has become the common practice of billion-scale kNN search (e.g., using the Faiss library [18]).

Example 3.2. Following Example 3.1, the text sequences for query and target columns are embedded to vectors by a fine-tuned PLM, as shown in Figure 2. The vectors for the 7 target columns are indexed in the approximate nearest neighbor (ANN) index. Given the vector of the query column, we search for its kNN ($k = 1$ in this example) by looking up the ANN index. Then the result is output as the joinable column, which refers to a target column in the repository.

3.4 Complexity Analysis

The search consists of two parts: query encoding and ANNS. In query encoding, we 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 $O(|M| \cdot |Q|)$, where $|M|$ denotes the model size. In ANNS, thanks to the use of HNSW, the time complexity [2] is $O(vl \log |\mathcal{X}|$, where v is the maximum out-degree (controlled by a parameter for index construction) in HNSW's index and l is the dimensionality of column embedding. Compared to JOSIE and PEXESO, which are linear in $|\mathcal{X}| \cdot (|Q| + \overline{|X|})$, we reduce the time complexity to sublinear in $|\mathcal{X}|$ and it is independent of $|Q|$ and $\overline{|X|}$ in the ANNS. Although the query encoding is still linear in $|Q|$, the column embedding procedure can be accelerated by GPUs.

[2] We follow the complexity analysis in [38].

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 it 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 examples. This can be done by a self-join on $\mathcal{X}$ 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 or use PEXESO for semantic joins. In case $\mathcal{X}$ is large, we can perform the self-join on a sample of $\mathcal{X}$.

In Definitions 2.1 and 2.3, the joinability is insensitive to the order of cells in a column, whereas PLMs are order-sensitive in their input. In order to make our model learn that the joinability is order-insensitive, we consider data augmentation by shuffling the cells in a 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 [33]. Given a batch of positive training examples $\{ (X_i, Y_i) \}$ (note that X_i may be a shuffled column), we assume each $(X_i, Y_j), Y_i \neq Y_j$ 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 shows better empirical results than other options of making negatives such as removing matching cells from positives.

4.2 Loss Function

Given a batch of N training examples $\{ (X_i, Y_i) \}$, we minimize the multiple negative ranking loss [25], which measures the negative log-likelihood 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 [44] for fine-tuning sentence-transformers [47]. For the scoring function $S(\cdot, \cdot)$, we choose the cosine similarity of column embeddings, which shows the best empirical results. The subtlety here is that in the top-k retrieval, Euclidean distance is used instead for the ANNS. The choice of metrics will be evaluated in Section 5.3.

Table 2: Dataset statistics.

Dataset	$\|X\|$	max. $\|X\|$	min. $\|X\|$	avg. $\|X\|$	# positive examples
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

5 EXPERIMENTS

5.1 Experimental Settings

Datasets. The following two datasets are used in the evaluation. (1) **Webtable** is a dataset of the WDC Web Table Corpus [48]. We use the English relational web tables 2015 and for each table, we extract the key column defined in the metadata. (2) **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 datasets contain metadata for table title, column name, and context, and have been used in previous works [2, 17, 56, 60, 64, 66]. Columns that are too short (< 5 cells) are removed. For semantic joins, fastText [19] is used to embed cells, Euclidean distance is used for distance function d, and the threshold τ for vector matching is 0.9, unless otherwise specified.

In order to show that DeepJoin learned from a small subset of a corpus is able to generalize to a large subset, we randomly sample two subsets of 30k and 1M columns, respectively, from each corpus. From the 30k training set, we randomly sample column pairs whose $jn \geq 0.7$ as initial positive examples, 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 1M testing set is used as the repository $\mathcal{X}$ for search. To generate queries and avoid data leaks, we randomly sample 50 columns from the original corpus except those in $\mathcal{X}$. The dataset statistics are given in Table 2.

Methods. We compare the following methods. (1) DeepJoin: This is our proposed model. We equip our model with DistilBERT [49] and MPNet [51] as PLM and denote the resultant model as $\text{DeepJoin}_{\text{DistilBERT}}$ and $\text{DeepJoin}_{\text{MPNet}}$, respectively. (2) JOSIE [64]: This is an exact solution to equi-joinable table discovery, based on top-k set similarity search. (3) LSH Ensemble [66]: This is an approximate solution to equi-joinable table discovery, based on partitioning and MinHash. (4) fastText, BERT, MPNet: We replace the column embedding in DeepJoin by averaging (no fine-tuning) the word embeddings from fastText [19], BERT [14], and MPNet [51], respectively. (5) TaBERT [60]: 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. (6) TURL [13]: This is a representation learning approach for table understanding tasks. We use its column embedding to replace that in DeepJoin. (7) 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 output of the last hidden layer as column embedding. (8) PEXESO [17]: This is an exact solution to semantic-joinable table discovery, using pivot-based filtering and a grid index.

Metrics. For accuracy, we evaluate precision@k and normalized discounted cumulative gain (NDCG@k). Precision@k measures the overlap between the model's top-k results and the top-k of an exact solution to Problem 1. NDCG@k is defined as $\frac{DCG_{\text{model}}}{DCG_{\text{exact}}}$, 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_{exact} are the top-k of the model and the exact solution, respectively. For semantic joins, we also request our colleagues of database researchers to label whether a retrieved table is really joinable, and then measure precision, recall, and F1 score. Precision = (# retrieved joinable columns) / (# retrieved columns). Since it is too laborious to label every table in the dataset, we follow [30] and build a retrieved pool using the union of the tables identified by the compared methods, which is also common practice for the evaluation of Web search engines. Recall = (# retrieved joinable columns) / (# joinable columns in the retrieved pool), where joinable columns are labeled by our experts. For efficiency, we evaluate the end-to-end processing time, including column-to-text transformation, query embedding, and ANNS. The above measures are averaged over all the queries.

Environments. DeepJoin are implemented with PyTorch. we use the Sentence-BERT [46] and the Hugging Face [1] libraries to build and train the DeepJoin model. We use the following setting: batch size = 32, learning rate = 2e-5, warmup steps = 10000, and weight decay rate = 0.01. Like DeepJoin, other column embedding methods (fastText, BERT, MPNet, TaBERT, TURL, and MLP) follow the same ANNS scheme, for which we use IVFPQ [31] and HNSW [38] in the Faiss library [18]. Experiments are run on a server with a 2.20GHz Intel Xeon CPU E7-8890 and 630 GB RAM. Models are (optionally) accelerated using a 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 always outperforms alternatives and exhibits outstanding generalizability (trained on 30k columns and tested on 1M columns). The best performance, with an average precision of 72% and NDCG of 81%, is observed when MPNet is equipped. $\text{DeepJoin}_{\text{MPNet}}$ is better than $\text{DeepJoin}_{\text{DistilBERT}}$ because MPNet is pre-trained on a larger corpora and under a unified view of masked language modeling and permuted language modeling. LSH Ensemble's performance is mediocre due to the conversion from overlap condition to Jaccard condition, which becomes very loose when the sizes of query and target significantly differ. For embedding methods, TURL is better than TaBERT because the pre-trained tasks (column type annotation, etc.) of TURL are closer to joinable table discovery than TaBERT's question answering. Nonetheless, these tasks still significantly differ from joinable table discovery, and thus both are in general no better than fastText and BERT. Another reason why TaBERT and TURL exhibit inferior performance is due to the limited data for pre-training; e.g., TURL is pre-trained on entity-focused Wikipedia tables. fastText is better than BERT and MPNet, indicating that simply using PLMs without fine-tuning does not translate to higher accuracy than context-insensitive word embeddings. MLP roughly performs the best among the methods other than DeepJoin, showing that a regression on top of word embeddings further improves the performance.

Table 3: Accuracy of equi-joins.

	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
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
TURL	0.653	0.669	0.689	0.711	0.721	0.688	0.706	0.716	0.727	0.732
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
TURL	0.504	0.525	0.529	0.545	0.578	0.707	0.711	0.745	0.766	0.778
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, $\tau = 0.9$ (labeled by PEXESO [17]).

	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
Webtable										
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$_{\text{DistilBERT}}$ (ours)	0.861	0.926	0.951	0.961	0.966	0.610	0.622	0.641	0.676	0.671
DeepJoin$_{\text{MPNet}}$ (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$_{\text{DistilBERT}}$ (ours)	0.788	0.835	0.876	0.880	0.913	0.803	0.807	0.810	0.826	0.831
DeepJoin$_{\text{MPNet}}$ (ours)	**0.813**	**0.881**	**0.889**	**0.889**	**0.936**	**0.814**	**0.820**	**0.833**	**0.842**	**0.852**

For semantic join, Table 4 reports the precision and NDCG evaluated under PEXESO's definition (Definition 2.3). DeepJoin$_{\text{MPNet}}$ reports an average precision of 91% and NDCG of 75%, delivering higher accuracy than alternatives for all the settings. fastText is competitive on Webtable but is not good on Wikitable. We also change the threshold τ for vector matching to 0.8 and 0.7, and report the accuracy in Tables 5 and 6, respectively. DeepJoin$_{\text{MPNet}}$ is still the best for low τ settings, though its precision and NDCG generally drop with τ. Such trend is also observed in most other methods. This is because a lower τ suggests that more cell values are regarded as matching, and thus it tends to introduce less similar contents to the training examples, which are harder to deal with.

We then evaluate these methods using the labels from our database researchers. The precision, recall, and F1 score when $k = 10$ are reported in Table 7. DeepJoin$_{\text{MPNet}}$ 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 score. We believe there are two reasons. First, DeepJoin$_{\text{MPNet}}$ 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 with a threshold. When judged by experts for joinability, the matching condition may differ across cell values, queries, and target columns, whereas a fixed threshold may not fit all of them. For a detailed comparison, we show three typical win/lose examples in Table 8. "Win" means only DeepJoin is able to correctly identify this entry, while "lose" means a false positive for DeepJoin but a true negative for at least one other competitor. The first example, which pertains to headboards, shows that the attention mechanism focuses on the word "headboard" in the table title and the words indicating bed size in the cell values. The second example, which pertains to tall buildings, shows that the PLM captures the similarity between the titles of the query and the target, as well as the building names in their contents. The third example, a false positive of DeepJoin, is potentially due to incorrect alignment of

Table 5: Accuracy of semantic joins, $\tau = 0.8$ (labeled by PEXESO [17]).

	Precision@k					NDCG@k				
				Webtable						
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
LSH Ensemble	0.571	0.592	0.621	0.613	0.633	0.604	0.613	0.622	0.628	0.636
fastText	0.551	0.561	0.565	0.599	0.614	0.597	0.619	0.618	0.625	0.621
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.734	0.746	0.776	0.831	0.850	0.621	0.637	0.676	0.699	0.704
DeepJoin$_{\text{MPNet}}$ (ours)	**0.774**	**0.791**	**0.823**	**0.845**	**0.881**	**0.655**	**0.684**	**0.723**	**0.729**	**0.737**
				Wikitable						
LSH Ensemble	0.499	0.529	0.497	0.491	0.504	0.573	0.570	0.569	0.573	0.582
fastText	0.395	0.480	0.523	0.549	0.607	0.203	0.204	0.210	0.222	0.223
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.621	0.714	0.758	0.776	0.811	0.598	0.632	0.676	0.688	0.703
DeepJoin$_{\text{MPNet}}$ (ours)	**0.659**	**0.758**	**0.803**	**0.805**	**0.846**	**0.620**	**0.670**	**0.694**	**0.710**	**0.722**

Table 6: Accuracy of semantic joins, $\tau = 0.7$ (labeled by PEXESO [17]).

	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
				Webtable						
LSH Ensemble	0.321	0.368	0.389	0.394	0.390	0.333	0.338	0.355	0.364	0.377
fastText	0.397	0.505	0.594	0.663	0.722	0.352	0.370	0.384	0.422	0.440
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.411	0.509	0.601	0.673	0.738	0.359	0.381	0.396	0.433	0.461
DeepJoin$_{\text{MPNet}}$ (ours)	**0.426**	**0.527**	**0.604**	**0.679**	**0.742**	**0.363**	**0.388**	**0.411**	**0.435**	**0.471**
				Wikitable						
LSH Ensemble	0.310	0.346	0.336	0.351	0.342	0.474	0.477	0.470	0.467	0.470
fastText	0.093	0.140	0.190	0.230	0.256	0.058	0.067	0.075	0.082	0.086
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.431	0.497	0.523	0.554	0.575	0.601	0.607	0.611	0.626	0.624
DeepJoin$_{\text{MPNet}}$ (ours)	**0.476**	**0.539**	**0.568**	**0.593**	**0.604**	**0.623**	**0.627**	**0.631**	**0.646**	**0.647**

Table 7: Accuracy of semantic joins, $\tau = 0.9$ (labeled by experts).

Methods	Precision	Recall	F1
	Webtable		
LSH Ensemble	0.181	0.228	0.202
fastText	0.138	0.277	0.183
PEXESO	0.212	0.506	0.300
DeepJoin$_{\text{MPNet}}$ (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$_{\text{MPNet}}$ (ours)	**0.842**	**0.568**	**0.677**

metadata. While these examples suggest that PLMs perform better than context-insensitive word embeddings when there are phrases indicating strong joinability, we also observe opportunities for improvement.

To drill down the cases of semantic joins, we randomly sample 100 query columns from the original corpus and request our experts to label the search results ($k = 10$) of four methods, LSH Ensemble, fastText, PEXESO, and DeepJoin$_{\text{MPNet}}$. We divide the results into two cases: joins for data cleaning (near duplicates) references and joins for related columns (attribute enrichment). In Webtable, there are 68 tables for near duplicates and 177 tables for attribute enrichment. In Wikitable, there are 164 tables for near duplicates and 330 tables for attribute enrichment. We report the recalls of the four competitors in Table 9. In general, lower recalls show that attribute enrichment is harder than near duplicates. This is expected, because for attribute enrichment, the matching condition is looser, meaning that we need to consider more columns that can be joined in a semantic manner. Nonetheless, DeepJoin$_{\text{MPNet}}$ exhibits superior performance in both cases, and the gap to the runner-up competitors are remarkable, especially for near duplicates, wherein the recall is around twice as much as the runner-up's.

To investigate how the performance changes with column size, we divide target columns of Webtable into three groups according to their size: short (5 – 10 cells), medium (11 – 50 cells), and long (> 50 cells). We only perform this experiment on Webtable because the number of columns in the long group is too small on Wikitable. For each group, we ensure that the query length is in the same range, and report the results in Table 10. For all the methods, the accuracy decreases with column size. This is because each column is transformed to a fixed-length object (MinHash sketch or vector). From the information perspective, the object after transformation has redundancy for short columns, but is compressed and more lossy for long columns. Nonetheless, DeepJoin$_{\text{MPNet}}$ is always the best, in line with what we have witnessed in the above experiments.

We also perform an experiment on a synthetic dataset with 512 to 4,096 rows, in order to investigate the case when the input sequence length exceeds the max_seq_length limit of PLMs (e.g., 512 tokens for DeepJoin). We synthesize 10k columns of 5 attributes: address, company, job, person, and profile, by using Faker [20] with the

Table 8: Win/lose examples for DeepJoin$_{\text{MPNet}}$ v.s. non-DeepJoin competitors on semantic joins labeled by experts.

Query	DeepJoin$_{\text{MPNet}}$'s result	Win/Lose	Possible reason
`title`: Headboard Buying Guide `colname`: Mattress Size `col`: California King, Full/Double, King, Queen, Twin, ...	`title`: Carved Headboard | west elm `colname`: Item `col`: Carved Headboard Full, Carved Headboard King, ...	Win	The attention mechanism in DeepJoin$_{\text{MPNet}}$ focuses on the word "Headboard" and bed size words such as "King" and "Double".
`title`: Tallest buildings `colname`: Name `col`: City Tower, City-Haus, City-Hochhaus Leipzig, ...	`title`: Buildings above 140m `colname`: Name `col`: Centrum LIM, City-Haus, City-Hochhaus, ...	Win	The PLM in DeepJoin$_{\text{MPNet}}$ captures the semantic similarity between "Tallest buildings" and "Buildings above 140m" in titles, as well as tall building names in cell values.
`title`: How to call Kazakhstan from Korea South `colname`: City `col`: Aktubinsk, Almaty, Arkalyk, ...	`title`: How to call Georgia from Georgia `colname`: City `col`: Akhalgori, Akhmeta, Aspindza, ...	Lose	The PLM in DeepJoin$_{\text{MPNet}}$ pays too much attention to metadata, but ignores that there are no similar values between the column contents.

Table 9: Recall@10, semantic joins drill-down: near duplicates (ND) and attribute enrichment (AE).

	Webtable		Wikitable	
Methods	ND	AE	ND	AE
LSH Ensemble	0.320	0.381	0.344	0.274
fastText	0.195	0.262	0.373	0.334
PEXESO	0.355	0.399	0.215	0.335
DeepJoin$_{\text{MPNet}}$	**0.701**	**0.561**	**0.649**	**0.418**

Table 10: Effect of varying column size.

	Precision@10			NDCG@10		
Methods	$\|X\| = 5-10$	$10-50$	>50	$\|X\| = 5-10$	$10-50$	>50
Webtable, equi-joins						
LSH Ensemble	0.647	0.633	0.617	0.722	0.693	0.688
fastText	0.692	0.694	0.673	0.764	0.751	0.719
BERT	0.684	0.663	0.642	0.755	0.731	0.714
MPNet	0.627	0.619	0.614	0.718	0.698	0.699
TaBERT	0.652	0.651	0.649	0.724	0.731	0.702
TURL	0.678	0.667	0.645	0.729	0.744	0.715
MLP	0.695	0.691	0.664	0.765	0.755	0.701
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.724	0.711	0.703	0.777	0.768	0.761
DeepJoin$_{\text{MPNet}}$ (ours)	**0.765**	**0.741**	**0.737**	**0.789**	**0.773**	**0.764**
Webtable, semantic joins						
LSH Ensemble	0.722	0.721	0.714	0.621	0.618	0.605
fastText	0.851	0.841	0.837	0.613	0.622	0.616
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.878	0.851	0.849	0.645	0.640	0.638
DeepJoin$_{\text{MPNet}}$ (ours)	**0.884**	**0.871**	**0.856**	**0.677**	**0.655**	**0.651**

Table 11: Evaluation on tall columns, synthetic, equi-joins.

	Precision@10				NDCG@10			
Methods	$\|X\| = 512$	1024	2048	4096	$\|X\| = 512$	1024	2048	4096
LSH Ensemble	0.518	0.521	0.505	0.511	0.611	0.606	0.597	0.593
fastText	0.577	0.568	0.561	0.563	0.634	0.639	0.624	0.615
BERT	0.578	0.554	0.545	0.547	0.635	0.622	0.621	0.600
MPNet	0.545	0.531	0.537	0.534	0.622	0.615	0.611	0.601
TaBERT-random	0.523	0.521	0.517	0.529	0.619	0.610	0.606	0.592
TURL-random	0.534	0.529	0.511	0.521	0.627	0.612	0.601	0.587
MLP	0.581	0.569	0.563	0.571	0.639	0.638	0.626	0.617
DeepJoin$_{\text{DistilBERT}}$-frequency (ours)	0.664	0.647	0.622	0.617	0.677	0.668	0.657	0.644
DeepJoin$_{\text{MPNet}}$-frequency (ours)	**0.697**	**0.671**	**0.666**	**0.669**	**0.696**	**0.674**	**0.677**	**0.665**
DeepJoin$_{\text{MPNet}}$-random (ours)	0.684	0.669	0.643	0.641	0.691	**0.674**	0.664	0.649
DeepJoin$_{\text{MPNet}}$-truncate (ours)	0.681	0.657	0.645	0.642	0.683	0.655	0.661	0.654

default parameters that match real-world English word frequencies. We randomly take 50 columns as queries and the others are targets. The method that samples the most frequent cell values within max_seq_length tokens (see Section 3.2), is dubbed -frequency. For comparison, we consider another two options: -random, which randomly samples cell values with no more than max_seq_length tokens, and -truncate, which truncates to the first max_seq_length tokens. The results are reported in Table 11. DeepJoin$_{\text{MPNet}}$ still consistently outperforms other models. For the three sampling options, -random is generally better than -truncate, and -frequency is always the best, justifying our argument that frequent cell values are more likely to yield join results.

5.3 Ablation Study

We first evaluate the impact of column-to-text transformation and test the seven options in Table 1. The results are reported in Tables 12 and 13. Adding column name at the beginning (`colname-col`) improves the performance of simply concatenating cell values (`col`). 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-stat-col` is the best.

We then evaluate the impact of cell shuffle for data augmentation. We vary the shuffle rate (defined in Section 4.1) and report the results in Tables 14 and 15. `0.0` means there is no shuffle. We observe that a moderate shuffle rate achieves the best performance (0.2 and 0.3 for equi-joins and semantic joins on Webtable, respectively, and 0.3 and 0.4 for equi-joins and semantic joins on Wikitable, respectively), indicating that shuffling the cells in columns helps the model learn that the 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 can 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.

For the column embedding metrics used in offline training and online searching, we perform an evaluation of two metrics: cosine similarity and Euclidean distance. Since the Faiss library does not support cosine similarity for ANNS, we normalize column embeddings before ANNS and use the inner product as the metric, so as to output the same kNN results as using cosine similarity. The precisions when $k = 10$ are reported in Table 19. Cosine similarity is better for training, while Euclidean distance is better for searching. The combination of cosine similarity for training and Euclidean distance for searching is overall the best. Since the performances of the two metrics are very close, users may address the discrepancy by choosing the same metric for training and searching. Nonetheless, we still use the best combination in our experiments.

5.4 Efficiency Evaluation

We vary the number of target columns and report the average query processing time in Table 16. 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 (e.g., around 2 times when

Table 12: Evaluation of column-to-text transformation, equi-joins.

	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
Webtable										
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-stat-col	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-stat-col	**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-stat-col	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-stat-col	**0.614**	**0.622**	**0.641**	**0.666**	**0.678**	**0.821**	**0.824**	**0.830**	**0.833**	**0.833**

Table 13: Evaluation of column-to-text transformation, semantic joins.

	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
Webtable										
col	0.826	0.833	0.866	0.885	0.925	0.610	0.615	0.623	0.637	0.644
colname-col	0.831	0.840	0.877	0.899	0.945	0.616	0.620	0.631	0.644	0.652
colname-col-context	0.831	0.839	0.875	0.886	0.945	0.620	0.631	0.640	0.650	0.661
colname-stat-col	0.834	0.846	0.887	0.904	0.956	0.625	0.641	0.659	0.654	0.671
title-colname-col	0.851	0.879	0.904	0.926	0.959	0.633	0.651	0.667	0.670	0.675
title-colname-col-context	0.850	0.877	0.915	0.927	0.954	0.631	0.650	0.671	0.675	0.677
title-colname-stat-col	**0.874**	**0.934**	**0.954**	**0.963**	**0.970**	**0.640**	**0.657**	**0.664**	**0.685**	**0.680**
Wikitable										
col	0.773	0.810	0.837	0.845	0.891	0.791	0.803	0.807	0.822	0.825
colname-col	0.775	0.815	0.842	0.847	0.903	0.797	0.807	0.811	0.829	0.834
colname-col-context	0.774	0.812	0.841	0.847	0.901	0.794	0.807	0.810	0.830	0.833
colname-stat-col	0.784	0.820	0.850	0.855	0.913	0.804	0.811	0.815	0.833	0.841
title-colname-col	0.804	0.836	0.868	0.874	0.922	0.811	0.815	0.821	0.837	0.844
title-colname-col-context	0.803	0.835	0.868	0.877	0.923	0.811	0.817	0.826	0.840	0.845
title-colname-stat-col	**0.813**	**0.881**	**0.889**	**0.889**	**0.936**	**0.814**	**0.820**	**0.833**	**0.842**	**0.852**

we increase the Webtable 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 (with MPNet), even if equipped with a CPU, is 7 – 57 times (Webtable) and 3 – 32 times (Wikitable) faster than the above methods. The growth of search time is also slight (e.g., 1.09 times for equi-joins and 1.05 times for semantic joins, with Webtable's size from 1M to 5M), showcasing its scalability. With the help of a GPU, DeepJoin is substantially accelerated and can be 103 and 421 times faster than JOSIE and PEXESO, respectively, and even faster than fastText.

Table 17 reports the query processing time when we vary k from 10 to 50. The general trend is that we spend more time for a larger k. Nonetheless, the growth for DeepJoin is very slight, because most of its overhead is query encoding, which is independent of the choice of k. As such, we roughly observe a greater speedup over existing methods when we increase k from 10 to 50.

To evaluate how the efficiency changes with column size, we use the same setting as in the corresponding accuracy evaluation (Table 10). Additionally, we sample and index only 300k target columns for each group, in order to eliminate the impact of the number of target columns. The results are reported in Table 18. The exact methods, JOSIE and PEXESO, exhibit considerable growth (1.9 and 1.5 times, respectively) of query processing time when we switch from short to long columns, which reflects the analysis in Section 2.2. In contrast, the growth for embedding methods is much slighter. For example, we only observe a growth of 1.09 times for DeepJoin with a CPU, and this only affects its query encoding rather than the ANNS. For DeepJoin with a GPU, we also observe a more remarkable speedup over the exact methods on longer columns.

Table 14: Evaluation of cell shuffle, equi-joins.

	Precision@k					NDCG@k				
shuffle rate	k = 10	20	30	40	50	k = 10	20	30	40	50
Webtable										
0.0	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										
0.0	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**	**0.664**	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 15: Evaluation of cell shuffle, semantic joins.

	Precision@k					NDCG@k				
shuffle rate	k = 10	20	30	40	50	k = 10	20	30	40	50
Webtable										
0.0	0.868	0.917	0.950	0.954	0.959	0.631	0.649	0.651	0.677	0.679
0.1	0.870	0.919	0.949	0.959	0.963	0.633	0.651	0.655	0.679	0.683
0.2	0.872	0.922	0.950	0.961	0.966	0.639	0.655	0.659	0.681	**0.687**
0.3	**0.874**	**0.934**	**0.954**	**0.963**	**0.970**	**0.640**	**0.657**	**0.664**	**0.685**	0.680
0.4	0.871	0.930	0.939	0.961	0.968	0.631	0.654	0.654	0.683	0.686
0.5	0.863	0.919	0.945	0.955	0.961	0.632	0.649	0.648	0.679	0.681
Wikitable										
0.0	0.798	0.856	0.865	0.877	0.914	0.801	0.804	0.813	0.820	0.833
0.1	0.801	0.861	0.870	0.881	0.921	0.803	0.806	0.819	0.822	0.839
0.2	0.806	0.866	0.875	0.883	0.925	0.806	0.810	0.822	0.825	0.840
0.3	0.808	0.870	0.877	0.887	0.929	0.809	0.812	0.825	0.826	0.843
0.4	**0.813**	**0.881**	**0.889**	**0.889**	**0.936**	**0.814**	**0.820**	**0.833**	**0.842**	**0.852**
0.5	0.809	0.871	0.873	0.880	0.931	0.809	0.813	0.829	0.829	0.844

6 RELATED WORK

Table discovery and data lake management. Besides joinable table discovery [64, 66], techniques have been developed for searching unionable tables [43]. Another important problem is related table discovery. SilkMoth [12] models columns as sets and finds related sets under maximum bipartite matching metrics. JUNEAU [63] finds related tables for data science notebooks using a composite score of multiple similarities. D^{3}L [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. Nextia$_{JD}$ [21] uses meta-features (cardinalities, value distribution, entropy, etc.) to provide a join quality ranking of candidate columns. EMBER [53] is a context enrichment system for ML pipelines, leveraging transformer-based [55] representation learning to automate keyless joins. Another notable work is Valentine [36], in which an experiment suite was proposed for the experiments of dataset discovery with joinability and unionability.

For data lake management, another problem which has been extensively studied is column type annotation. Notable approaches include Sherlock [28], Sato [61], and DODUO [52]. 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 [65] joins two tables with string transformations on columns. A similar method is auto-transform [24], which learns string transformations with patterns. Besides, SEMA-join [23] finds related pairs between two tables with statistical correlation. Other representative problems include data lake organization [42], data validation [50], and data lake integration [34]. A recent advancement is regarding data integration as prompting tasks for foundation models [41].

Table embedding. Language models have been used in understanding the contents of tables. For example, cell classification was investigated in [22], with an RNN-based cell embedding method proposed. Table2Vec [62], featuring a series of embedding approaches for words, entities, and headers based on the idea of skip-gram model of word2vec [40], deals with the table retrieval problem that

Table 16: Processing time per query, varying $|\mathcal{X}|$, $k = 10$.

	query encoding (ms)	total (ms)				
Methods		$\|\mathcal{X}\|$ = 1M	2M	3M	4M	5M
Webtable, equi-joins						
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 joins						
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**
		$\|\mathcal{X}\|$ = 200k	400k	600k	800k	1M
Wikitable, equi-joins						
LSH Ensemble	-	236	338	467	514	652
JOSIE	-	304	377	455	556	647
fastText	6	6.7	6.6	6.8	6.9	7.4
DeepJoin (CPU)	76	76.4	76.7	76.9	77.0	77.1
DeepJoin (GPU)	**5**	**5.4**	**5.8**	**5.8**	**6.1**	**6.3**
Wikitable, semantic joins						
PEXESO	1665	1874	1995	2310	2551	2789
DeepJoin (CPU)	86	86.5	86.9	87.1	87.4	87.7
DeepJoin (GPU)	**9**	**9.5**	**9.6**	**10.1**	**10.3**	**10.5**

Table 17: Processing time per query, varying k.

	query encoding (ms)	total (ms)				
Methods		k = 10	20	30	40	50
Webtable, equi-joins						
LSH Ensemble	-	496	506	590	595	508
JOSIE	-	535	556	578	580	506
fastText	9	10.3	10.5	10.2	10.8	11.1
DeepJoin (CPU)	66	67.1	67.1	67.1	67.2	68.1
DeepJoin (GPU)	7	8.4	8.1	8.2	8.1	8.0
Webtable, semantic joins						
PEXESO	-	2345	2444	2356	2754	2566
DeepJoin (CPU)	74	75.6	76.8	76.1	75.8	76.1
DeepJoin (GPU)	7	8.1	8.3	8.0	8.2	8.4
Wikitable, equi-joins						
LSH Ensemble	-	652	720	715	678	736
JOSIE	-	647	667	708	697	788
fastText	6	7.4	7.2	7.8	7.3	7.7
DeepJoin (CPU)	76	77.1	78.1	77.4	77.5	77.6
DeepJoin (GPU)	5	6.3	7.0	6.6	6.7	6.4
Wikitable, semantic joins						
PEXESO	-	2655	2776	2557	2743	2789
DeepJoin (CPU)	86	87.4	87.3	87.1	87.2	87.7
DeepJoin (GPU)	9	10.5	11	10.2	10.7	10.4

returns a ranked list of tables for a keyword query. Besides, PLMs such as BERT [14] have also been used for table retrieval [9].

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 them. TURL [13] features a contextualization technique to convert table contents to sequences and leverages a masked language model (MLM) initialized by TinyBERT [32]. TaPas [26] is built upon a similar MLM but employs BERT [14], with additional information embedded such as positions and ranks. TaBERT [60] pre-trains for question answering tasks and learns embeddings for cells, columns, and utterance tokens in the questions using BERT. By adopting two transformers to independently encode rows and columns, TABBIE [29] embeds cells, columns, and rows, and achieves one order of magnitude less training time than TaBERT. TUTA [57] creates trees to encode the information in hierarchical tables, while most previous studies focused on flat tables. Another method for hierarchical tables is GTR [56], which models cells, rows, and columns as nodes in a graph and employs a graph transformer [35] to capture the neighborhood information of cells, rows, and columns.

Table 18: Processing time per query, varying $|X|$, $k = 10$.

	query encoding (ms)			total (ms)		
Methods	$\|X\|$ = 5 − 10	11 − 50	> 50	$\|X\|$ = 5 − 10	11 − 50	> 50
Webtable, equi-joins						
LSH Ensemble	-	-	-	455	487	467
JOSIE	-	-	-	410	589	792
fastText	5	6	6	5.8	6.7	6.9
DeepJoin (CPU)	71	75	78	71.7	75.4	78.5
DeepJoin (GPU)	4	5	6	4.9	5.8	6.9
Webtable, semantic joins						
PEXESO	-	-	-	2123	2785	3244
DeepJoin (CPU)	81	84	89	81.9	84.7	89.3
DeepJoin (GPU)	8	9	9	8.8	9.6	10.0

Table 19: Evaluation of embedding metrics, precision@10.

	$\text{DeepJoin}_{\text{DistilBERT}}$		$\text{DeepJoin}_{\text{MPNet}}$	
Webtable, equi-joins				
train \ search	cosine	Euclidean	cosine	Euclidean
cosine	0.701	**0.702**	0.730	0.731
Euclidean	0.697	0.700	0.732	**0.732**
Wikitable, equi-joins				
train \ search	cosine	Euclidean	cosine	Euclidean
cosine	**0.588**	**0.588**	**0.614**	**0.614**
Euclidean	0.587	0.585	0.611	0.612
Webtable, semantic joins				
train \ search	cosine	Euclidean	cosine	Euclidean
cosine	**0.861**	**0.861**	0.870	**0.874**
Euclidean	0.859	0.859	0.870	0.866
Wikitable, semantic joins				
train \ search	cosine	Euclidean	cosine	Euclidean
cosine	0.787	**0.788**	0.810	**0.813**
Euclidean	0.785	0.783	0.808	0.810

7 CONCLUSION

We proposed DeepJoin, a deep learning model that fits both equi- and semantic-joinable table discovery in a data lake. DeepJoin was designed in an embedding-based retrieval fashion, which embeds columns with a fine-tuned PLM and resorts to ANNS to find joinable results, thereby achieving a search time sublinear in the repository size. The experiments demonstrated the generalizability of DeepJoin, which is consistently more accurate than alternatives methods, as well as the superiority of DeepJoin in search speed, which is up to two orders of magnitude faster than alternatives. We expect that by employing more advanced retrieval strategies or PLMs, the performance of DeepJoin can be further improved.

ACKNOWLEDGMENTS

This work is mainly supported by NEC Corporation and partially supported by JSPS Kakenhi 22H03903 and CREST JPMJCR22M2.

REFERENCES

[1] Hugging face transformers. https://huggingface.co/docs/transformers/index, 2022.

[2] C. S. Bhagavatula, T. Noraset, and D. Downey. Tabel: Entity linking in web tables. In *ISWC*, volume 9366 of *Lecture Notes in Computer Science*, pages 425–441. Springer, 2015.

[3] C. S. Bhagavatula, T. Noraset, and D. Downey. Wikitables. http://websail-fe.cs.northwestern.edu/TabEL/, 2015.

[4] S. Bharadwaj, P. Gupta, R. Bhagwan, and S. Guha. Discovering related data at scale. *PVLDB*, 14(8):1392–1400, 2021.

[5] A. Bogatu, A. A. A. Fernandes, N. W. Paton, and N. Konstantinou. Dataset discovery in data lakes. In *ICDE*, pages 709–720, 2020.

[6] A. Z. Broder. On the resemblance and containment of documents. In *SEQUENCES*, pages 21–29. IEEE, 1997.

[7] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5. IEEE Computer Society, 2006.

[8] L. Chen, Y. Gao, B. Zheng, C. S. Jensen, H. Yang, and K. Yang. Pivot-based metric indexing. *PVLDB*, 10(10):1058–1069, 2017.

[9] Z. Chen, M. Trabelsi, J. Heflin, Y. Xu, and B. D. Davison. Table search using a deep contextualized language model. In *SIGIR*, pages 589–598. ACM, 2020.

[10] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger. ARDA: automatic relational data augmentation for machine learning. *PVLDB*, 13(9):1373–1387, 2020.

[11] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *RecSys*, pages 191–198. ACM, 2016.

[12] D. Deng, A. Kim, S. Madden, and M. Stonebraker. Silkmoth: An efficient method for finding related sets with maximum matching constraints. *PVLDB*, 10(10):1082–1093, 2017.

[13] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu. TURL: table understanding through representation learning. *PVLDB*, 14(3):307–319, 2020.

[14] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186. ACL, 2019.

[15] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[16] Y. Dong and M. Oyamada. Table enrichment system for machine learning. In *SIGIR*, pages 3267–3271. ACM, 2022.

[17] Y. Dong, K. Takeoka, C. Xiao, and M. Oyamada. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *ICDE*, pages 456–467. IEEE, 2021.

[18] facebook AI research. Faiss: Facebook ai similarity search. https://github.com/facebookresearch/faiss/wiki/Faiss-indexes, 2022.

[19] Facebook AI Research Lab. fastText: Library for efficient text classification and representation learning. https://fasttext.cc/, 2015.

[20] D. Faraglia. Faker. https://github.com/joke2k/faker, 2023.

[21] J. Flores, S. Nadal, and O. Romero. Effective and scalable data discovery with nextiajd. In *EDBT*, pages 690–693. OpenProceedings.org, 2021.

[22] M. Ghasemi-Gol, J. Pujara, and P. A. Szekely. Tabular cell classification using pre-trained cell embeddings. In *ICDM*, pages 230–239. IEEE, 2019.

[23] Y. He, K. Ganjam, and X. Chu. SEMA-JOIN: joining semantically-related tables using big table corpora. *PVLDB*, 8(12):1358–1369, 2015.

[24] Y. He, Z. Jin, and S. Chaudhuri. Auto-transform: Learning-to-transform by patterns. *PVLDB*, 13(11):2368–2381, 2020.

[25] M. L. Henderson, R. Al-Rfou, B. Strope, Y. Sung, L. Lukács, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil. Efficient natural language response suggestion for smart reply. *CoRR*, abs/1705.00652, 2017.

[26] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. In *ACL*, pages 4320–4333. ACL, 2020.

[27] J. Huang, A. Sharma, S. Sun, L. Xia, D. Zhang, P. Pronin, J. Padmanabhan, G. Ottaviano, and L. Yang. Embedding-based retrieval in facebook search. In *KDD*, pages 2553–2561. ACM, 2020.

[28] M. Hulsebos, K. Z. Hu, M. A. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, Ç. Demiralp, and C. A. Hidalgo. Sherlock: A deep learning approach to semantic data type detection. In *KDD*, pages 1500–1508. ACM, 2019.

[29] H. Iida, D. Thai, V. Manjunatha, and M. Iyyer. TABBIE: pretrained representations of tabular data. In *NAACL-HLT*, pages 3446–3456. ACL, 2021.

[30] C. S. J. and W. Peter. Estimating the recall performance of web search engines. *Aslib Proceedings*, 49(7):184–189, Jan 1997.

[31] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.

[32] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. Tinybert: Distilling BERT for natural language understanding. In *EMNLP (Findings)*, volume EMNLP 2020 of *Findings of ACL*, pages 4163–4174. ACL, 2020.

[33] V. Karpukhin, B. Oguz, S. Min, P. S. H. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih. Dense passage retrieval for open-domain question answering. In *EMNLP*, pages 6769–6781. ACL, 2020.

[34] A. Khatiwada, R. Shraga, W. Gatterbauer, and R. J. Miller. Integrating data lake tables. *PVLDB*, 16(4):932–945, 2022.

[35] R. Koncel-Kedziorski, D. Bekal, Y. Luan, M. Lapata, and H. Hajishirzi. Text generation from knowledge graphs with graph transformers. In *NAACL-HLT*, pages 2284–2293. ACL, 2019.

[36] C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, J. Brons, M. Fragkoulis, C. Lofi, A. Bonifati, and A. Katsifodimos. Valentine: Evaluating matching techniques for dataset discovery. In *ICDE*, pages 468–479. IEEE, 2021.

[37] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan. Deep entity matching with pre-trained language models. *PVLDB*, 14(1):50–60, 2020.

[38] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, 2020.

[39] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[40] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[41] A. Narayan, I. Chami, L. J. Orr, and C. Ré. Can foundation models wrangle your data? *PVLDB*, 16(4):738–746, 2022.

[42] F. Nargesian, K. Q. Pu, E. Zhu, B. G. Bashardoost, and R. J. Miller. Organizing data lakes for navigation. In *SIGMOD*, pages 1939–1950. ACM, 2020.

[43] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *PVLDB*, 11(7):813–825, 2018.

[44] Nils Reimers. Sentencetransformers.losses. https://www.sbert.net/docs/package_reference/losses.html, 2019.

[45] J. Qin, W. Wang, C. Xiao, Y. Zhang, and Y. Wang. High-dimensional similarity query processing for data science. In *KDD*, pages 4062–4063. ACM, 2021.

[46] N. Reimers. Sentence bert. https://www.sbert.net/, 2022.

[47] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP-IJCNLP*, pages 3980–3990. ACL, 2019.

[48] D. Ritze, O. Lehmberg, R. Meusel, C. Bizer, and S. Zope. WDC web table corpus. http://webdatacommons.org/webtables/2015/downloadInstructions.html, 2015.

[49] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[50] J. Song and Y. He. Auto-validate: Unsupervised data validation using data-domain patterns inferred from data lakes. In *SIGMOD*, pages 1678–1691. ACM, 2021.

[51] K. Song, X. Tan, T. Qin, J. Lu, and T. Liu. Mpnet: Masked and permuted pre-training for language understanding. In *NeurIPS*, 2020.

[52] Y. Suhara, J. Li, Y. Li, D. Zhang, Ç. Demiralp, C. Chen, and W. Tan. Annotating columns with pre-trained language models. In *SIGMOD*, pages 1493–1503. ACM, 2022.

[53] S. Suri, I. F. Ilyas, C. Ré, and T. Rekatsinas. Ember: No-code context enrichment via similarity-based keyless joins. *PVLDB*, 15(3):699–712, 2021.

[54] N. Tang, J. Fan, F. Li, J. Tu, X. Du, G. Li, S. Madden, and M. Ouzzani. RPT: relational pre-trained transformer is almost all you need towards democratizing data preparation. *PVLDB*, 14(8):1254–1261, 2021.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[56] F. Wang, K. Sun, M. Chen, J. Pujara, and P. A. Szekely. Retrieving complex tables with multi-granular graph representation learning. In *SIGIR*, pages 1472–1482. ACM, 2021.

[57] Z. Wang, H. Dong, R. Jia, J. Li, Z. Fu, S. Han, and D. Zhang. TUTA: tree-based transformers for generally structured table pre-training. In *KDD*, pages 1780–1790. ACM, 2021.

[58] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. H. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022.

[59] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3):15:1–15:41, 2011.

[60] P. Yin, G. Neubig, W. Yih, and S. Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. In *ACL*, pages 8413–8426. ACL, 2020.

[61] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, Ç. Demiralp, and W. Tan. Sato: Contextual semantic type detection in tables. *PVLDB*, 13(11):1835–1848, 2020.

[62] L. Zhang, S. Zhang, and K. Balog. Table2vec: Neural word and entity embeddings for table population and retrieval. In *SIGIR*, pages 1029–1032. ACM, 2019.

[63] Y. Zhang and Z. G. Ives. Finding related tables in data lakes for interactive data science. In *SIGMOD*, pages 1951–1966, 2020.

[64] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *SIGMOD*, pages 847–864. ACM, 2019.

[65] E. Zhu, Y. He, and S. Chaudhuri. Auto-join: Joining tables by leveraging transformations. *PVLDB*, 10(10):1034–1045, 2017.

[66] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. LSH ensemble: Internet-scale domain search. *PVLDB*, 9(12):1185–1196, 2016.