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 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 joins. 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 logarithmic 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:
[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

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 [57, 59], 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 [59], 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 [57].

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. To resolve the efficiency issue, DeepJoin 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 [35], whose time complexity is *logarithmic* 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 [48] 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

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

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. 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.
- We design the search in DeepJoin as an embedding-based retrieval which employs a PLM for column embedding and ANNS for fast retrieval. The search time complexity is logarithmic in the table repository size, and except the column embedding, the search time is independent of the column size.
- We propose a prompt engineering method to transform column contents to a text sequence fed to the PLM.
- 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.
- We conduct experiments to show that our model, featuring embedding-based retrieval with a PLM, 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 [57]. 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].

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 [57] 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

[1] Besides this definition, semantic joins are also investigated in [22], yet it studies the problem of performing joins rather than searching for joinable columns.

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* $\min\{\, jn(Q,X) \mid X \in \mathcal{R} \,\} \geq jn(Q,Y), \forall Y \in \mathcal{X} \backslash \mathcal{R}$.

2.2 State-of-the-Art

JOSIE [57], 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 [52], 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 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 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. 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 [59] 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 [57].

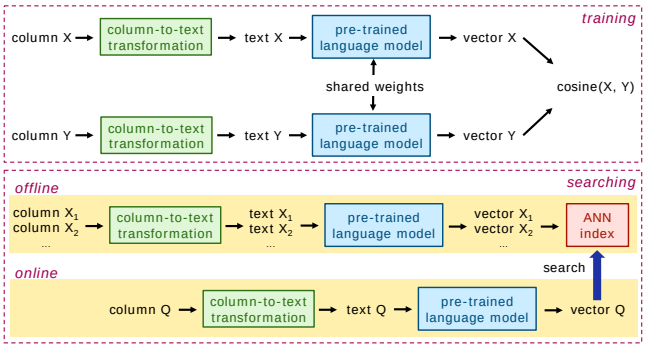

Figure 1: Overview of the DeepJoin model.

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 [32] and search in social networks [26]. 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 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 [47], entity matching [34], and column annotation [46]) 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 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 PLM to produce a column embedding.

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	$table_title$. $colname-col-stat$.

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 [51]. 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 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 [43], a faster and lighter variant of BERT [14], and MPNet [45], 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 [41] to output a sentence embedding for a sequence of input text. In case of a limit of the input sequence's length (e.g., 512 tokens), we choose the cell values with the highest frequency (i.e., the number of target columns containing the cell value for the column-to-text transformation. 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 PLM.

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) [35], which is among the most prevalent approaches to ANNS [40]. Since the search time complexity of HNSW is claimed to be logarithmic in the number of indexed objects [35], 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) [30], 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]).

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, due to the use of HNSW, the time complexity 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 logarithmic 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.

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

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

text retrieval tasks [32]. 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 [24], which measures the negative log-likelihood of softmax normalized scores:

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

The above loss function is one of the prevalent options [39] for fine-tuning sentence-transformers [41]. 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. We believe the difference is because that the length of embeddings is also useful in identifying joinable results.

5 EXPERIMENTS

5.1 Experimental Settings

Datasets. The following datasets are used in the evaluation.

- **Webtable** is a dataset of the WDC Web Table Corpus [42]. 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, 17, 49, 53, 57, 59]. 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.

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 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 equip our model with DistilBERT [43] and MPNet [45] as PLM and denote the resultant model as $\text{DeepJoin}_{\text{DistilBERT}}$ and $\text{DeepJoin}_{\text{MPNet}}$, respectively.
- JOSIE [57]: This is an exact solution to equi-joinable table discovery, based on top-k set similarity search.
- LSH Ensemble [59]: This is an approximate solution to equi-joinable table discovery, based on partitioning and MinHash.
- 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 [45], respectively.
- TaBERT [53]: 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 [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 top-k results output by the model and the top-k output by 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 results of the model and the exact solution, respectively.

To evaluate the effectiveness of 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 tables) / (# retrieved tables). Since it is too laborious to label every table in the dataset, we follow [29] and build a retrieved pool using the union of the tables identified by the compared methods. 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 ANNS.

Accuracy and efficiency are reported by averaging over all the queries used in the experiments.

Environments. DeepJoin are implemented with PyTorch, the Hugging Face Transformers library [1], and the Sentence-BERT library [41]. We use the following setting: batch size = 32, learning rate = 2e-5, warmup steps = 10000, and weight decay rate = 0.01. Like DeepJoin, other methods involving column embedding (fastText, BERT, MPNet, TaBERT, and MLP) follow the same ANNS scheme, for which we use IVFPQ [30] and HNSW [35] in the Faiss library [18], as described in Section 3.3. 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 single NVidia A100 Tensor Core. All the competitors are implemented in Python 3.7.

Table 3: Accuracy of equi-joins.

Webtable										
	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
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, $\tau = 0.9$ (labeled by PEXESO [17]).

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$_{\text{DistilBERT}}$ (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	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**

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. DeepJoin$_{\text{MPNet}}$ is better than 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, fastText and BERT are generally better than TaBERT, because TaBERT is pre-trained for question answering which significantly differs from joinable table discovery. 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.

For semantic join, Table 4 reports the precision and NDCG evaluated under PEXESO's definition (Definition 2.3). For the performance of DeepJoin, DeepJoin$_{\text{MPNet}}$ still consistently outperforms DeepJoin$_{\text{DistilBERT}}$. 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 other methods. This is expected, because a low τ setting 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 are reported in Table 7. DeepJoin$_{\text{MPNet}}$

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

	Webtable									
	Precision@k					NDCG@k				
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]).

	Webtable									
	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
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 (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$_{\text{MPNet}}$ (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**

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 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.

To investigate how the accuracy 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). For each group, we ensure that the query length is in the same range, and report the results in Table 8. It can be seen that for all the methods, the accuracy, as shown in precision and NDCG, decreases with column size. This is because each column is transformed to a fixed-length object (a MinHash sketch or a 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 is always the best among all the competitors, and MPNet is still the better PLM for DeepJoin, in line with what we have witnessed in the above experiments.

Table 8: Accuracy, varying column size, Webtable, $k = 10$.

	Equi-joins					
	Precision@k			NDCG@k		
Methods	$\|X\| = 5-10$	$10-50$	>50	$\|X\| = 5-10$	$10-50$	>50
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
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**
	Semantic joins					
	Precision@k			NDCG@k		
Methods	$\|X\| = 5-10$	$10-50$	>50	$\|X\| = 5-10$	$10-50$	>50
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 9: Evaluation of column-to-text transformation, equi-joins.

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

	Webtable									
	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
`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**

5.3 Ablation Study

We first evaluate the impact of column-to-text transformation and test the seven options in Section 3.1. The results are reported in Tables 9 and 10. 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-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 11 and 12. 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.

5.4 Efficiency Evaluation

We vary the number of target columns and report the average query processing time in Table 13. For embedding methods, we

Table 11: Evaluation of cell shuffle, equi-joins.

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**	**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 12: Evaluation of cell shuffle, semantic joins.

Webtable										
	Precision@k					NDCG@k				
shuffle rate	k = 10	20	30	40	50	k = 10	20	30	40	50
no-shuffle	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										
no-shuffle	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

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 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, 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), showing the scalability of embedding methods. 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 14 reports the query processing time when we vary k from 10 to 50. As expected, 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 query processing time changes with column size, we divide the target columns of Webtable into three groups and ensure the query lengths are in the same range, in line with the corresponding accuracy evaluation (Table 8). Additionally, we sample and index only 300K target columns for each group, in order to eliminate the impact of the number of target columns in this experiment. The average query processing time is reported in Table 15. 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 complexity analysis in Section 2.2. In contrast, the time growth for query encoding 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 the GPU

Table 13: Processing time per query, varying $\mathcal{X}$, $k = 10$.

Methods	query encoding (ms)	total (ms) $\lvert\mathcal{X}\rvert$ = 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**
Methods	query encoding (ms)	total (ms) $\lvert\mathcal{X}\rvert$ = 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 14: Processing time per query, varying k.

Methods	query encoding (ms)	total (ms) k = 10	20	30	40	50
Webtable, equi-joins						
LSH Ensemble [59]	-	496	506	590	595	508
JOSIE [57]	-	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 [59]	-	652	720	715	678	736
JOSIE [57]	-	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

version of DeepJoin, we also observe a more remarkable speedup over the exact methods on longer columns.

6 RELATED WORK

Table discovery and data lake management. Besides joinable table discovery [57, 59], techniques have been developed for searching unionable tables [38]. Another important problem is related table discovery. SilkMoth [12] models columns as sets and finds related sets under maximum bipartite matching metrics. JUNEAU [56] 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. Another notable approach is Nextia$_{JD}$ [20],

Table 15: Processing time per query, varying column size, Webtable, $k = 10$.

Methods	query encoding (ms) $\lvert Q\rvert$ = 5 − 10	11 − 50	> 50	total (ms) $\lvert X\rvert$ = 5 − 10	11 − 50	> 50
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
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

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 [27], Sato [54], and DODUO [46]. 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 [58] joins two tables with string transformations on columns. A similar method is auto-transform [23], which learns string transformations with patterns. Besides, SEMA-join [22] finds related pairs between two tables with statistical correlation. Other representative problems of data lake managment include data lake organization [37] and data validation [44].

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 [21], with an RNN-based cell embedding method proposed. Table2Vec [55], which features a series of embedding approaches for words, entities, and headers based on the idea of skip-gram model of word2vec [36], deals with the table retrieval problem that returns a ranked list of tables for a keyword query. Besides, PLMs such as BERT [14] have also been used for table retrieval [9], accompanied by an embedding-based feature selection technique to select relevant contents to a query and a multi-layer perceptron 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 them. TURL [13] features a contextualization technique to convert table contents to sequences and leverages a masked language model (MLM) initialized by TinyBERT [31]. TaPas [25] is built upon a similar MLM but employs BERT [14], with additional information embedded such as positions and ranks. TaBERT [53] pre-trains for question answering tasks and learns embeddings for cells, columns, and utterance tokens in the questions using BERT. By adopting two transformers [48] to independently encode rows and columns, TABBIE [28] embeds cells, columns, and rows, and achieves one order of magnitude less training time than TaBERT. As a tree-based model using BERT, TUTA [50] creates trees to encode the information in hierarchical tables, while most previous studies focused on flat tables. Another method for hierarchical tables is GTR [49], which models cells, rows, and columns as nodes in a graph and employs a

graph transformer [33] to capture the neighborhood information of cells and the global information of rows and columns.

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 PLM and resorts to a state-of-the-art ANNS algorithm to find joinable results, thereby achieving a search time logarithmic in the repository size. A metric learning approach was proposed so that columns are expected to be joinable if they are close in the embedding space. We devised techniques for contextualization and training data preparation. The experiments showed the generalizability of DeepJoin, which is consistently more accurate than alternative methods and even better than an exact solution to semantic join when evaluated on expert-labeled data. The evaluation also showed 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.

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*. 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*. 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*. ACM, 589–598. https://doi.org/10.1145/3397271.3401044

[10] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *PVLDB* 13, 9 (2020), 1373–1387. http://www.vldb.org/pvldb/vol13/p1373-chepurko.pdf

[11] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*. ACM, 191–198. https://doi.org/10.1145/2959100.2959190

[12] 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

[13] 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

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

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

[16] Yuyang Dong and Masafumi Oyamada. 2022. Table Enrichment System for Machine Learning. In *SIGIR*. ACM, 3267–3271. https://doi.org/10.1145/3477495.3531678

[17] 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

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

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

[20] Javier Flores, Sergi Nadal, and Oscar Romero. 2021. Effective and Scalable Data Discovery with NextiaJD. In *EDBT*. OpenProceedings.org, 690–693. https://doi.org/10.5441/002/edbt.2021.85

[21] Majid Ghasemi-Gol, Jay Pujara, and Pedro A. Szekely. 2019. Tabular Cell Classification Using Pre-Trained Cell Embeddings. In *ICDM*. IEEE, 230–239. https://doi.org/10.1109/ICDM.2019.00033

[22] 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

[23] 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

[24] 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

[25] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *ACL*. Association for Computational Linguistics, 4320–4333. https://doi.org/10.18653/v1/2020.acl-main.398

[26] 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*. ACM, 2553–2561. https://doi.org/10.1145/3394486.3403305

[27] 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*. ACM, 1500–1508. https://doi.org/10.1145/3292500.3330993

[28] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *NAACL-HLT*. Association for Computational Linguistics, 3446–3456. https://doi.org/10.18653/v1/2021.naacl-main.270

[29] 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

[30] 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

[31] 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)*, Vol. EMNLP 2020. Association for Computational Linguistics, 4163–4174. https://doi.org/10.18653/v1/2020.findings-emnlp.372

[32] 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*. Association for Computational Linguistics, 6769–6781. https://doi.org/10.18653/v1/2020.emnlp-main.550

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

[34] 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

[35] 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

[36] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 3111–3119. https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[37] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. 2020. Organizing Data Lakes for Navigation. In *SIGMOD*. ACM, 1939–1950. https://doi.org/10.1145/3318464.3380605

[38] 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

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

[40] Jianbin Qin, Wei Wang, Chuan Xiao, Ying Zhang, and Yaoshu Wang. 2021. High-Dimensional Similarity Query Processing for Data Science. In *KDD*. ACM, 4062–4063. https://doi.org/10.1145/3447548.3470811

[41] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*. Association for Computational Linguistics, 3980–3990. https://doi.org/10.18653/v1/D19-1410

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

[43] 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

[44] Jie Song and Yeye He. 2021. Auto-Validate: Unsupervised Data Validation Using Data-Domain Patterns Inferred from Data Lakes. In *SIGMOD*. ACM, 1678–1691. https://doi.org/10.1145/3448016.3457250

[45] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. MPNet: Masked and Permuted Pre-training for Language Understanding. In *NeurIPS*. https://proceedings.neurips.cc/paper/2020/hash/c3a690be93aa602ee2dc0ccab5b7b67e-Abstract.html

[46] 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*. ACM, 1493–1503. https://doi.org/10.1145/3514221.3517906

[47] 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

[48] 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*. 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[49] Fei Wang, Kexuan Sun, Muhao Chen, Jay Pujara, and Pedro A. Szekely. 2021. Retrieving Complex Tables with Multi-Granular Graph Representation Learning. In *SIGIR*. ACM, 1472–1482. https://doi.org/10.1145/3404835.3462909

[50] 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*. ACM, 1780–1790. https://doi.org/10.1145/3447548.3467434

[51] 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

[52] 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

[53] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*. Association for Computational Linguistics, 8413–8426. https://doi.org/10.18653/v1/2020.acl-main.745

[54] 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

[55] Li Zhang, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *SIGIR*. ACM, 1029–1032. https://doi.org/10.1145/3331184.3331333

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

[57] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*. ACM, 847–864. https://doi.org/10.1145/3299869.3300065

[58] 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

[59] 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