TabSketchFM: Sketch-based Tabular Representation Learning for Data Discovery over Data Lakes *

Aamod Khatiwada[⋆▶]
Northeastern University
khatiwada.a@northeastern.edu

Harsha Kokel[⋆]
IBM Research
harsha.kokel@ibm.com

Ibrahim Abdelaziz
IBM Research
ibrahim.abdelaziz1@ibm.com

Subhajit Chaudhury
IBM Research
subhajit@ibm.com

Julian Dolby
IBM Research
subhajit@ibm.com

Oktie Hassanzadeh
IBM Research
hassanzadeh@us.ibm.com

Zhenhan Huang[▶]
Rensselaer Polytechnic Institute
huangz12@rpi.edu

Tejaswini Pedapati
IBM Research
tejaswinip@us.ibm.com

Horst Samulowitz
IBM Research
samulowitz@us.ibm.com

Kavitha Srinivas
IBM Research
kavitha.srinivas@ibm.com

Abstract

In this paper, we present TabSketchFM, a neural tabular model for data discovery over data lakes. First, we propose novel pre-training: a sketch-based approach to enhance the effectiveness of data discovery in neural tabular models. Second, we finetune the model for identifying unionable, joinable, and subset table pairs and show significant improvement over previous tabular neural models. Third, we use these finetuned models to perform table search; i.e., given a query table, find other tables in a corpus that are unionable, joinable, or that are subsets of the query. Our results demonstrate significant improvements for search compared to state-of-the-art techniques. Finally, we show significant transfer across datasets and tasks establishing that our model can generalize across different tasks and over different data lakes.

1 Introduction

Enterprises store critical data in *data lakes*, large repositories of tabular data, for both governance and analytics [22]. It is essential to effectively find relevant tables (e.g., joinable, unionable, subsets) within lakes for reporting, decision-making, statistical analysis, and more [19, 20]. For instance, *Unionable table search* helps to augment an existing table with new rows, and to enrich analytics over it [23]. *Joinable table* search is useful to run analytics that needs data in columns from multiple tables, e.g., to identify how many customers in some region purchased a product due to an email campaign. Also, identifying *subsets* or *supersets* of a table is essential, for example, when searching for potential copies of personal data to comply with regulations [2]. Motivated by such use cases, we focus on the problem of identifying tables from the data lakes that can be *unionable, joinable, and/or*

[*⋆] ⋆ Joint first-authors ▶ Work done while at IBM Research.

[2] https://gdpr-info.eu

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

subsets of each other [23, 30, 18]. Specifically, in this work, we propose a pre-trained tabular model that could help in enhancing the effectiveness of data discovery techniques.

Central to the problem of identifying relevant tables in the data lakes is identifying similar columns in tables. However, the columns in a table could be ambiguous. For instance, a column called `Age` is different when it is in a table about buildings from when it is in a table about people. Furthermore, values within a column `Age` play a key role in determining semantics; `Age` of buildings in New York are unlikely to match `Age` of historical buildings in Egypt. Contextualizing columns could be useful in resolving such ambiguities. Language Models (LMs) are shown to be successful in contextualizing words in a sentence [8]. Consequently, we use them to contextualize columns and use such contextualization to determine similar columns or tables.

Existing pre-trained tabular models [7, 16, 27, 28] based on LMs [8] do not focus on identifying relevant tables. Instead, they focus on language-style tasks such as fact checking [27], question answering [14], converting natural language to SQL [28], table cell content population [7], and table metadata prediction [16, 7]. In addition, input to these models is the actual cell values or a row because of the focus on language style tasks. However, this has two limitations for understanding relevant tables: (a) only a small number of values can be passed in as input because of severe limits in the length of input tokens, at least in most tabular models built so far; (b) more importantly, treating numerical values in the table as text tends to lose their semantics; it maybe better to pass numerical information directly as vectors into the model.

To address these limitations on input length and numerical representation, we introduce a novel transformer-based tabular model—*TabSketchFM*—exploiting both table metadata and cell values that creates different sketches over them instead of linearizing cell values. Specifically, we abstract column cell values using data sketches, as has been used in the data discovery literature (e.g., MinHash sketches [31], and numerical sketches [24]). Such sketches capture tabular features, and we input them into the neural models. Inputting numerical sketches bypasses feeding table cell values to the model, addressing the token limit issue, as well as the numerical representation of cell values.

Following a phase of pretraining with sketches, we fine tuned the models for the data discovery tasks of union, join, and finding subsets. To finetune models for the data discovery tasks, we used a suite of 8 benchmarks across the 3 tasks defined in *LakeBench*[26]. On the fine tuning tasks, *TabSketchFM* outperformed other neural methods, by as much as 55% on F1. We then used the fine tuned models for search tasks, and showed that *TabSketchFM* outperforms the state-of-the-art neural and traditional methods by as much as 70% on F1.

Importantly, we also compare the performance of specialized traditional and neural models for data discovery against an off-the-shelf SBERT model [3] that encodes column values as a single sentence. To our surprise, the off-the-shelf model performs much better than existing systems - in fact, for certain tasks such as union, it is sufficient to use that model for discovering unionable tables; our tabular specific model does worse, as do all other models that target unionability. The insight from this finding is that while some data discovery tasks need values to match across columns (e.g. join or subsets), for union, it is sufficient when columns have the same semantic meaning; indeed there is no need for any value overlap. Sentence embeddings capture those semantics well. Because certain tasks seem to benefit from these sentence embeddings (e.g. union), and other do not (e.g., subset), we concatenated the embeddings from sentence encoders with tabular model embeddings, and show that one can augment tabular embeddings with value embeddings to improve performance, at least in some cases.

In summary, our contributions are as follows:

- **A novel tabular pretrained model.** We present a sketch-based tabular pretrained model, *TabSketchFM*, which contextualizes columns based on their data sketches and tabular metadata. A purely sketch based approach though misses the opportunity to include table semantics from column values. Given that the inputs to the transformer architecture in our system are numeric vectors, it is difficult to combine these numeric inputs with tokens derived from cell values. We therefore combine the sketch based model embeddings with off the shelf embeddings of column values from existing pretrained sentence transformer models. We show that this combination boosts performance on some data discovery tasks.

[3] `https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-12-v2`

- **Demonstration of TabSketchFM on search.** We compared the performance of *TabSketchFM*'s fine tuned models against neural and non-neural baselines. In the case of join search, *TabSketchFM* outperformed the nearest systems geared for join search by almost 70% in F1 scores, but tellingly, the sentence encoding model outperformed existing systems as well by 63%. By combining the two embeddings (from TabSketchFM and sentence encodings), we gain almost 73% over other systems in F1.
- **Illustrate generalization.** Finally, we illustrate the main advantage of learning a pretrained foundational model by showing the generalization of search performance across tasks. For example, we fine tune the pretrained model on join identification on one dataset and use it for union search on another dataset. Our evaluation shows that our finetuned models generalize well.

2 Related Work

Tabular Pretrained Models. Neural models are pretrained on tabular datasets to either recover masked tokens in the table or to detect corrupt values [7, 16, 27, 28], see [1] for a survey neural models for tabular data representation and their applications. These models are finetuned and evaluated for different downstream tasks such as table metadata prediction, table content population, fact-checking, question answering, semantic parsing, and so on.

Dataset Discovery Methods. Different methods such as keyword search [4, 3] and table search [25, 21, 12] have been used for the dataset discovery over data lakes. Table Union Search (TUS) [23] presented finding top-k data lake tables that are unionable with a query table. They determine the column unionability using three statistical tests based on value overlap, semantic overlap, and word embedding similarity, and aggregate them to infer table unionability. D^3L [2], adopted value overlap and word embedding measures from TUS and added three additional measures to measure column unionability: numerical column distributions, column header similarity, and regular expression matching. SANTOS[18] considers the relationships between the column pairs in both query and data lake tables. [11] developed a contrastive-learning-based table union search technique called Starmie that captures the context of the whole table in the form of column embeddings. AutoTUS [15] searches for unionable tables by contextualizing entire tables with embeddings of the relationship between the column pairs. Furthermore, there are other table search techniques that focus on finding joinable tables, i.e., given a query table marked with a query column, find the data lake tables that contain a column with overlapping values with the query column. For instance, LSH Ensemble [31] defines joinable table search as a problem of finding data lake tables that have a column having high set containment with the query column. They propose a novel index that approximates set containment computation. JOSIE [30] searches for top-k joinable tables based on exact set containment. PEXESO [9] efficiently searches for the top-k semantically joinable tables. Along with a join on exact values, they also consider fuzzy join, i.e., join on abbreviations and synonyms. DeepJoin [10] is a deep-learning-based technique that also searches for semantically joinable tables. WarpGate [5] is also a system for searching for top-k semantically joinable tables by embedding each column to vector space using pre-trained embeddings. They index column embeddings using Locally Sensitive Hashing [13] for efficient search. Recently, Chorus [17] even demonstrated the ability of Large Language Models to identify joinable columns as a task of completing 'pd.merge' command when two tables are presented as pandas dataframe. Our work addresses the task of discovering tables from the data lakes that can be unionable, joinable, and/or subsets of each other. Notably, we specialize data discovery neural models based on a single pretrained model. The advantage of our approach over above mentioned task specific approaches is that, as in natural language processing, much smaller datasets are sufficient to finetune the model for new data discovery tasks; by leveraging the existing tabular knowledge in the pretrained model.

3 TabSketchFM

Like existing tabular models [27, 14], we pretrain a transformer-based BERT model [8], but change the architecture to accommodate numerical inputs. The goal of building the pretrained model is to create fine tuned encoders that can be used for ranking in data discovery using significantly smaller sized labeled datasets.

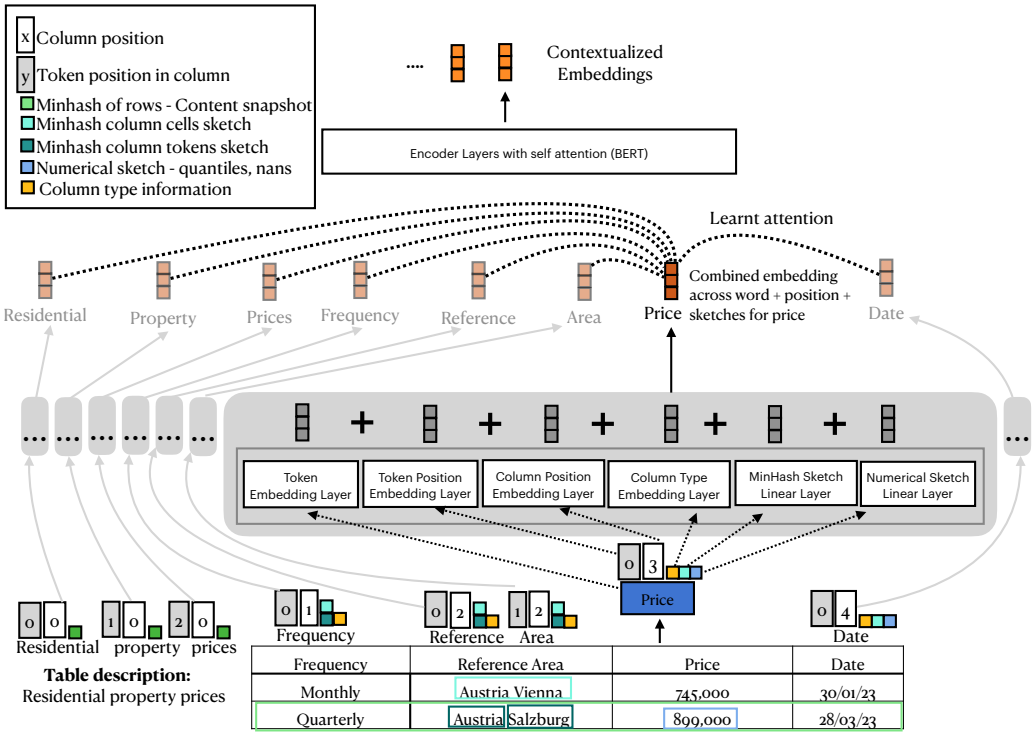


Figure 1: Incorporation of sketches into transformer architecture. MinHashes are computed over the whole cell value, and tokens of a cell value for a given column. MinHashes are also computed over the entire row, treated as a string. For a given column token (e.g., Price in this figure), hidden states for each encoding are summed after passing through an embedding or a linear layer before it gets sent to the BERT encoder layers. Table description tokens are summed with MinHashes over the entire row.

The first departure from other tabular representation models is in how we represent the table as input. While other models linearize table cells as text, we create *a sketch for each column* which are numerical vectors that need to be translated into a form that can be consumed by the transformer architecture. The sketches capture different features of the columns. Figure 1 shows an overview of how the different sketches we describe below are fed to the self-attention layers. Specifically, we create the following sketches for each column.

1. Numerical sketches. For each column, we record the number of NaNs and the number of unique values (normalized by the number of rows in the table). For string columns, we also record cell width in bytes, since that can be an important determinant in governing if the column is likely to be a join column. For instance, very long strings are unlikely to be join candidates. When possible, we convert date columns into timestamps and then treat them as numeric columns. For numeric columns, we additionally compute a percentile sketch, and add the mean, the standard deviation, and minimum and maximum values. All these features are encoded as individual elements of a single input vector we call the numerical sketch.

2. MinHash sketches. For each column, we compute a MinHash signature of cell values.[4] As shown in Fig. 1, the cell value of `Austria Vienna` is used as a single element in the set passed to the MinHash computation.[5] For string columns, we also compute another MinHash signature for the tokens within the column. The rationale for this second MinHash is that the tokens can sometimes capture its semantics (e.g., if a token `street` appears in two different columns, they maybe both about address, which is useful if one considers a union of two tables that do not necessarily share cell values). As shown in Fig. 1, the cell values of `Austria` and `Salzburg` get tokenized and used as

[4]In our implementation, we use datasketch library [29] to compute MinHash.

[5]While it does not make sense to convert float or integer columns into MinHash signatures, we do so because it is often difficult to tell if a column is truly a float or an integer, or it really is a categorical value.

two different elements of the set passed to the MinHash computation. We concatenate both MinHash sketches into a single input vector for string columns. For numerical and date columns, only the MinHash for the cell values is included in the input vector.

3. Content snapshot. Row-based information could be crucial in understanding the table semantics for data discovery [18]. We concatenate values within each row to form a string and create MinHash signature for each of them. For instance, we concatenate the last row of table in Fig. 1, surrounded by a green rectangle, into "*Quarterly Austria Salzburg 800,000 28/03/23*" and hash it as another minhash style sketch.

Now that we have these sketches, which are arrays of numbers, a key question is how to adapt BERT's architecture to consume these numeric values. Transformer models take sentences as input, and internally break it into different inputs such as the tokens of a sentence[6], the position of tokens in the sentence, and a scalar type indicating if the token belongs to the first sentence or the second. We leverage this mechanism to pass the table-specific scalar inputs as shown in Fig. 1 to the BERT Embedding Layers. For vector inputs (e.g., numerical sketch, MinHash sketch), we expand the notion of BERT Embeddings to add some linear layers, as described below.

TabSketchFM's specialized layers for processing tabular input are shown in the light gray box, middle-right of Fig. 1. The token `Price` for the `Price` column is used as an example for illustrative purposes. All other tokens belonging to the table description and other columns would undergo a similar process to create a single embedding per token.

1. Token Embedding Layer. The embedding layer of the BERT uncased model [8] contains numerical embeddings for each word in its vocabulary based on extensive training with a large corpus, such that, for instance, the embedding of `Price` might be closely associated with that of `House` because they co-occur often in text. Our model's token embedding layer is initialized with the weights of the BERT model's embedding layer to leverage such information present in natural language, but the weights are allowed to be changed further while pretraining to include co-occurrence information in table-specific corpora. As shown in Fig. 1, the token `Price` would be mapped from its position in the vocabulary of the model (also pre-populated with BERT's vocabulary) into its hidden state using weights from this token embedding layer.

2. Token Position Embedding Layer. Each column is analogous to a sentence in a paragraph. Hence, we re-purpose token positional embeddings to reflect a token's position within a column name. For instance, as shown by the light gray scalar input in Fig. 1, the token position of a token `Area` is 1 because it is the second token in the cell (after `Reference`).

3. Column Position Embedding Layer. Columns themselves have positions and are encoded through a new embedding layer which can range from 1 to the total number of columns,[7] as shown by the white scalars in Fig. 1. The rationale for including column positions is that they sometimes do have semantics; e.g., a street address next to the city, followed by a zipcode. Of course table semantics do not change as a function of column order; nevertheless, we included position in case it helps the model understand column semantics, with the assumption that the attentional mechanism would disregard order if necessary.

4. Column Type Embedding Layer. Column types of string, date, integer, or float are encoded through another embedding. In order to determine column type, we made a best-case effort to parse the first 10 values of each column as dates, integers, or floats and defaulted to string if we could not convert them. In mixed-type columns, this can yield poor results, but at the very worst, at least one of the types was assigned to these columns. Column types are shown in Fig. 1 in orange.

5. MinHash Sketch Linear Layer. MinHash sketches and the content snapshots were passed through a linear layer to encode them into the same hidden state dimensions of the existing transformer layers in Bert.

6. Numerical Sketch Linear Layer. Numerical sketches are similarly passed through a linear layer to encode them.

[6]The BERT tokenizer sometimes breaks a single word into multiple tokens such that unknown words in its vocabulary are handled correctly.

[7]Position 0 is reserved for any tokens in the table description.

As shown in Fig. 1, for each column token, we combined the hidden states of its token embedding, token position embedding, column position embedding, column type embedding, MinHash sketch embedding, and numerical sketch embedding using summation. This summation is similar to the summation of token embeddings and position embeddings in the original BERT model. Content snapshot signatures are summed with table description tokens because the content snapshot describes the entire table's content. For instance, for the tokens `Residential`, `Property` and `Prices`, the content snapshot is passed into the linear layers to create a hidden state that is combined with the other embeddings for those tokens. Once each token has been summed with its other sketches, positional and type encoding, it is passed to the rest of the encoder and attention layers from BERT. The attention layers are crucial for contextualizing the embeddings based on what other columns, values, and table descriptions exist in the table. As shown in the figure for `Price`, the attention mechanism learns N^2 attention weights between all N tokens for a given table. This step will consider `Price` in relation to all other columns as well as table descriptions. We used the usual 12-layer BERT encoder model with self attention from HuggingFace[8] once we have unified the hidden states from the embedding layers and the linear layers.

4 Pretraining

Dataset. We created a de-duplicated pretraining dataset of $197,254$ enterprise-like tables (CSVs) from CKAN and Socrata that are legally cleared to have the open licenses. This pretrain dataset contains 2234.5 rows and 35.8 columns in each table on average. About 66% of columns were non-string, resembling an enterprise datalake.

Data Augmentation. We want the model to be robust to the order of columns such that shuffling columns in a table does not impact its semantics (see [6] for requirements for table embeddings). So, we created three different versions of the table, by changing the column order, which in turn, changed the table's content snapshot (see Section 3).

Method. We employ the standard Mask Language Modeling (MLM) as the pretraining objective for our model, as in most text-based language models. In our scenario, for each table, we mask a single column name such that all tokens corresponding to that column are masked. This is analogous to the natural language literature's whole word masking where all tokens corresponding to a word are masked. In addition, we used MLM probability to mask the tokens in the table description as well.

For large tables with more than 5 columns, we randomly select five columns and mask them. Following this strategy over the augmented pretraining dataset, we get $730,553$ examples in training, $54,430$ examples in validation, and $58,141$ examples in test sets. We provide the experimental setup for pretraining in Section 6.

5 Fine-tuning

Table 1 summarizes the main characteristics of the datasets to show the type of tasks we used for fine tuning (the first 8 tasks are from [26]). We use cross-entropy loss for classification tasks, mean squared error for regression tasks, and binary cross-entropy with logit loss for multi-class classification tasks. The goal of building multiple encoders for the same task across different datasets was to examine whether the encoders would generalize across datasets; allowing re-use to new data with no additional training.

6 Experiments

6.1 Finetuning results

We compare TabSketchFM against *four* publicly available tabular foundational models: TUTA, TaBERT, TABBIE, and TAPAS. We adapted these models for Lakebench data discovery tasks by building a dual encoder architecture. Each encoder represents the pretrained model with shared parameters; and the same model is used to encode each element of a table pair. The embeddings

[8] `https://huggingface.co/docs/transformers/model_doc/bert`

Table 1: Cardinality of all the datasets in LakeBench, as well as search benchmarks in this paper.

Benchmark	Task	# Tables	Avg. Rows	Avg. Cols	# Table Pairs			Data type distribution (%)			
					Train	Test	Valid	String	Int.	Float	Date
TUS-SANTOS	Binary Classification	1127	4285.17	13.04	16592	3566	3552	77.94	8.62	7.51	5.93
Wiki Union	Binary Classification	40752	51.05	2.66	301108	37638	37638	57.97	14.38	24.25	3.4
ECB Union	Regression	4226	292.47	36.3	15344	1910	1906	47.72	14.05	36.31	1.92
Wiki Jaccard	Regression	8489	47.3	2.8	12703	1412	1572	57.5	15.66	19.76	7.07
Wiki Containment	Regression	10318	47.15	2.79	21007	2343	2593	57.26	15.26	20.58	6.9
Spider-OpenData	Binary Classification	10730	1208.87	9.09	5146	742	1474	42.22	18.51	32.62	6.66
ECB Join	Multi-label Clasification	74	8772.24	34.47	1780	222	223	52.14	7.8	37.79	2.27
CKAN Subset	Binary Classification	36545	1832.58	25.37	24562	2993	3010	31.75	17.53	46.14	4.58
Eurostat Subset	Search	38904	2157	10.46				64.63	9.03	7.83	18.50
Wikijoin	Search	46521	49.64	2.68				58.13	13.35	25.0	3.50

Table 2: Performance (avg $\pm$ stdev) of TabSketchFM vs. other baseline models on LakeBench averaged across five random seeds. **Best** performance is highlighted in bold, and the second best is underlined.

Tasks	Vanilla BERT	TAPAS	TABBIE	TUTA	TaBERT	TabSketchFM (ours)
TUS-SANTOS (F1)	$\mathbf{0.99 \pm 0.00}$	0.34 ± 0.00	$\underline{0.75 \pm 0.22}$	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.99 \pm 0.00}$
Wiki Union (F1)	0.33 ± 0.00	0.41 ± 0.00	0.64 ± 0.12	0.33 ± 0.00	$\mathbf{0.97 \pm 0.00}$	$\underline{0.94 \pm 0.00}$
ECB Union (R2)	0.03 ± 0.03	-0.01 ± 0.01	0.02 ± 0.01	$\underline{0.87 \pm 0.01}$	0.35 ± 0.06	$\mathbf{0.90 \pm 0.03}$
Wiki Jaccard (R2)	0.00 ± 0.00	-0.03 ± 0.03	0.25 ± 0.02	$\underline{0.43 \pm 0.02}$	0.33 ± 0.03	$\mathbf{0.58 \pm 0.03}$
Wiki Containment (R2)	0.00 ± 0.00	0.00 ± 0.00	0.21 ± 0.01	$\underline{0.35 \pm 0.03}$	0.30 ± 0.03	$\mathbf{0.58 \pm 0.02}$
Spider-OpenData (F1)	0.71 ± 0.06	0.65 ± 0.00	0.57 ± 0.00	0.76 ± 0.12	$\mathbf{0.87 \pm 0.02}$	$\underline{0.83 \pm 0.01}$
ECB Join (F1)	0.63 ± 0.04	0.40 ± 0.00	0.42 ± 0.03	$\underline{0.81 \pm 0.01}$	0.79 ± 0.03	$\mathbf{0.86 \pm 0.01}$
CKAN Subset (F1)	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	$\underline{0.43 \pm 0.00}$	$\mathbf{0.98 \pm 0.00}$

from the last layer of the encoders were concatenated and passed through a two-layered MLP (multi-layered perceptron). While we were able to finetune the TUTA and TaBERT code in this dual encoder architecture, the code of TAPAS and TABBIE were less amenable to finetuning. Hence, for TAPAS and TABBIE, we froze their pretrained models while finetuning, but allowed the two layers above the model to learn the weights. In this case, the pretrained model can be seen as producing an embedding that is then fed into two layered network that tunes for a specific task. We will make the code for all the baselines available.

We also added a naive baseline, **Vanilla BERT**, to examine to what extent LakeBench tasks can be performed based on column headers alone; this provides a measure of the difficulty of a given task. For *Vanilla BERT*, column headers were provided for the two tables as two sentences, and the BERT model was finetuned on the specific LakeBench task as were other models.

Table 2 compares the performance of TabSketchFM with baseline models on LakeBench. For regression tasks, we report R2 statistics, and for (binary and multiclass) classification tasks, we report a weighted F1 score to handle skew in classes[9]. The best performance is highlighted in bold and the second best is underlined. All the results presented are aggregated over 5 random seeds. We see that on *five of the eight tasks, TabSketchFM outperformed all baselines, performed as well as the best methods on one, and performed second best on the remaining two*.

6.2 Application to search

We evaluate the embeddings from our finetuned TabSketchFM for three data discovery tasks: join search, union search, and subset search on the search benchmarks of TUS, SANTOS (for union), Wikijoin (for join) and Eurostat (for subsets). We developed the latter two which we developed due to the lack of existing subset benchmarks (see benchmark characteristics in Table 1). The new search benchmarks will be made publicly available.

For *join search*, we compare against state-of-the-art table join search techniques: traditional approaches such as **LSHForest** and **Josie** [30]—and three neural methods–**WarpGate** [5], **DeepJoin** [10], as well as the fine tuned version of **TaBERT**, and the **SBERT** baseline. TUTA was not

[9]Our metrics were implemented from `scikit-learn`: `https://scikit-learn.org/stable/modules/model_evaluation.html`

Table 3: Search across all benchmarks

(a)

Baseline	Mean F1	P@10	R@10
Wikijoin			
TaBERT-FT	5.88	0.43	0.04
LSH-Forest	10.48	0.8	0.08
Josie	19.56	**0.99**	0.12
DeepJoin	18.88	0.96	0.11
WarpGate	18.58	0.95	0.11
SBERT	83.67	0.95	0.89
TabSketchFM	89.09	0.97	0.94
TabSketchFM-SBERT	**92.81**	0.98	**0.99**
SANTOS			
TaBERT-FT	36.64	0.63	0.46
TUTA-FT	25.34	0.43	0.3
Starmie	**54.08**	**0.97**	0.72
D3L	26.44	0.54	0.4
SANTOS	50.36	0.89	0.67
SBERT	53.86	**0.97**	**0.73**
TabSketchFM	51.38	0.92	0.69
TabSketchFM-SBERT	**54.09**	**0.97**	**0.73**

(b)

Baseline	Mean F1	P@10	R@10
TUS			
TaBERT-FT	26.66	0.90	0.30
TUTA-FT	27.27	0.89	0.31
Starmie	27.48	0.96	0.32
D3L	18.98	0.75	0.21
SANTOS	20.83	0.81	0.23
SBERT	**31.13**	**0.99**	**0.36**
TabSketchFM	30.43	0.97	0.35
TabSketchFM-SBERT	30.72	**0.99**	0.35
Eurostat			
TABERT-FT	4.03	0.05	0.05
TUTA-FT	9.82	0.13	0.12
SBERT	43.12	0.56	0.51
TabSketchFM	**49.96**	**0.59**	**0.53**
TabSketchFM-SBERT	47.54	0.58	0.52

added because it only has table embeddings, so it could not be used to find joinable columns. For *union search*, we compare against state-of-the-art table union search techniques: $\mathbf{D^3L}$ [2], **SANTOS** [18], and **Starmie** [11] as well as the finetuned version of **TaBERT** and **TUTA**. Plus, an **SBERT** baseline explained earlier. For *subset search*, we could only compare against the fine tuned versions of **TaBERT** and **TUTA**.

Table 3 shows the significant advantage of TabSketchFM over all other baselines when considering F1 scores, and furthermore, adding column embeddings using SBERT improved performance for joins. Only for TUS (a union task) did SBERT outperform all other systems. In that case, combining TabSketchFM with SBERT was the second best system.

We also investigated if the models encoded on specific datasets or tasks generalize to different search applications. Because the core operation is determining column similarity in data discovery, generalization maybe expected. A change in the task or dataset showed at most a 2% drop in accuracy, showing that the fine tuned models generalize rather well across tasks as well as domain. We present a representative result of transfering TabSketchFM finetuned for different tasks and evaluated for Eurostat subset search in Fig. 2. The subset search seemed to benefit from training on union or join search, with the best performing models being models trained on similar datasets (data from the European central bank) but different tasks (join and union).

7 Conclusion and Discussion

We presented TabSketchFM, a novel transformer-based tabular representation model that inputs the table in the form of several sketches. We show that cross encoders based on TabSketchFM outperform existing systems on LakeBench related table identification task. We also show that finetuned TabSketchFM can be effectively used to search for tables that are unionable, joinable or are subsets of each other. We highlight that embedding values using sentence encoder models can be a useful addition for some discovery tasks. As far as the search efficiency is concerned, we recommend indexing the datalake offline and at query time only compute embeddings for the query table. This computation is similar to other embedding-based search system.

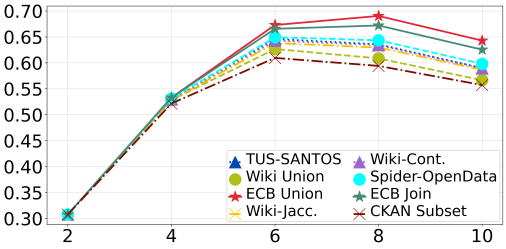


Figure 2: F1 on *Eurostat* transfer on search for varying k.

References

[1] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. Transformers for Tabular Data Representation: A Survey of Models and Applications. *Transactions of the Association for Computational Linguistics*, 11:227–249, 03 2023.

[2] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. Dataset discovery in data lakes. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 709–720, 2020.

[3] Dan Brickley, Matthew Burgess, and Natasha Noy. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *The World Wide Web Conference*, page 1365–1375, 2019.

[4] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549, 2008.

[5] Tianji Cong, James Gale, Jason Frantz, H. V. Jagadish, and Çagatay Demiralp. Warpgate: A semantic join discovery system for cloud data warehouses. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*, 2023.

[6] Tianji Cong, Madelon Hulsebos, Zhenjie Sun, Paul Groth, and H. V. Jagadish. Observatory: Characterizing embeddings of relational tables. *Proc. VLDB Endow.*, 17(4):849–862, 2023.

[7] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: table understanding through representation learning. *Proc. VLDB Endow.*, 14(3):307–319, 2020.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

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

[10] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. Deepjoin: Joinable table discovery with pre-trained language models. *Proc. VLDB Endow.*, 16(10):2458 – 2470, 2023.

[11] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *Proc. VLDB Endow.*, 16(7):1726–1739, 2023.

[12] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. Aurum: A data discovery system. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 1001–1012, Paris, France, 2018.

[13] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 518–529, 1999.

[14] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4320–4333, 2020.

[15] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and Philip S. Yu. Automatic table union search with tabular representation learning. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 3786–3800, 2023.

[16] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. TABBIE: pretrained representations of tabular data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 3446–3456, 2021.

[17] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. Chorus: Foundation models for unified data discovery and exploration. *Proc. VLDB Endow.*, 17(8):2104–2114, may 2024.

[18] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. Santos: Relationship-based semantic table union search. *Proc. ACM Manag. Data*, 1(1):Article 9, 2023.

[19] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J. Miller. Integrating data lake tables. *Proc. VLDB Endow.*, 16(4):932–945, 2022.

[20] Aamod Khatiwada, Roee Shraga, and Renée J. Miller. DIALITE: discover, align and integrate open data tables. In *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 187–190, 2023.

[21] Oliver Lehmberg and Christian Bizer. Stitching web tables for improving matching quality. *Proc. VLDB Endow.*, 10(11):1502–1513, 2017.

[22] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. Data lake management: Challenges and opportunities. *Proc. VLDB Endow.*, 12(12):1986–1989, 2019.

[23] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. Table union search on open data. *Proc. VLDB Endow.*, 11(7):813–825, 2018.

[24] Aécio S. R. Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. Correlation sketches for approximate join-correlation queries. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1531–1544, 2021.

[25] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 817–828, 2012.

[26] Kavitha Srinivas, Julian Dolby, Ibrahim Abdelaziz, Oktie Hassanzadeh, Harsha Kokel, Aamod Khatiwada, Tejaswini Pedapati, Subhajit Chaudhury, and Horst Samulowitz. Lakebench: Benchmarks for data discovery over data lakes, 2023.

[27] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. TUTA: tree-based transformers for generally structured table pre-training. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 1780–1790, 2021.

[28] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8413–8426, 2020.

[29] Eric Zhu and Vadim Markovtsev. ekzhu/datasketch: First stable release, February 2017.

[30] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 847–864, 2019.

[31] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. LSH ensemble: Internet-scale domain search. *Proc. VLDB Endow.*, 9(12):1185–1196, 2016.

A Supplemental Material

We attach a longer version of the full paper and appendix below.

TabSketchFM: Sketch-based Tabular Representation Learning for Data Discovery over Data Lakes

***Abstract*—Enterprises have a growing need to identify relevant tables in data lakes; e.g. tables that are unionable, joinable, or subsets of each other. Tabular neural models can be helpful for such data discovery tasks. In this paper, we present TabSketchFM, a neural tabular model for data discovery over data lakes. First, we propose novel pre-training: a sketch-based approach to enhance the effectiveness of data discovery in neural tabular models. Second, we finetune the pretrained model for identifying unionable, joinable, and subset table pairs and show significant improvement over previous tabular neural models. Third, we present a detailed ablation study to highlight which sketches are crucial for which tasks. Fourth, we use these finetuned models to perform table search; i.e., given a query table, find other tables in a corpus that are unionable, joinable, or that are subsets of the query. Our results demonstrate significant improvements in F1 scores for search compared to state-of-the-art techniques. Finally, we show significant transfer across datasets and tasks establishing that our model can generalize across different tasks and over different data lakes.**

I. Introduction

Enterprises store critical data in *data lakes*, large repositories of tabular data, for both governance and analytics [1]. It is essential to effectively find relevant tables (e.g., joinable, unionable, subsets) within lakes for reporting, decision-making, statistical analysis, and more [2], [3]. For instance, *Unionable table search* helps to augment an existing table with new rows, and to enrich analytics over it [4]. *Joinable table* search is useful to run analytics that needs data in columns from multiple tables, e.g., to identify how many customers in some region purchased a product due to an email campaign. Also, identifying *subsets* or *supersets* of a table is essential, for example, when searching for potential copies of personal data to comply with the General Data Privacy Regulation of the European Union[1]. Motivated by such use cases, we focus on the problem of identifying tables from the data lakes that can be *unionable, joinable, and/or subsets of each other* [4]–[6]. Specifically, in this work, we propose a pre-trained tabular model that could help in enhancing the effectiveness of data discovery techniques.

Central to the problem of identifying relevant tables in the data lakes is identifying similar columns in tables. However, the columns in a table could be ambiguous. For instance, a column called `Age` is different when it is in a table about buildings from when it is in a table about people. Furthermore, values within a column `Age` play a key role in determining semantics; `Age` of buildings in New York are unlikely to match `Age` of historical buildings in Egypt. Contextualizing columns could be useful in resolving such ambiguities. Language Models (LMs) are shown to be successful in contextualizing words in a sentence [7]. Consequently, we use them to contextualize columns and use such contextualization to determine similar columns or tables.

Existing pre-trained tabular models [8]–[11] based on LMs [7] do not focus on identifying relevant tables. Instead, they focus on language-style tasks such as fact checking [10], question answering [12], converting natural language to SQL [11], table cell content population [8], and table metadata prediction [8], [9]. In addition, input to these models is the actual cell values or a row because of the focus on language style tasks. However, this has two limitations for understanding relevant tables: (a) only a small number of values can be passed in as input because of severe limits in the length of input token (e.g. 512 tokens in BERT [7]), and this potentially limits the information available to the model to understand the input table; (b) treating numerical values in the table as text tends to lose their semantics; it maybe better to pass numerical information directly as vectors into the model.

To address these limitations on input length and numerical representation, we introduce a novel transformer-based tabular model—*TabSketchFM*—exploiting both table metadata and cell values that creates different sketches over them instead of linearizing cell values. Specifically, we abstract column cell values using data sketches, as has been used in the data discovery literature (e.g., MinHash sketches [13], and numerical sketches [14]). Such sketches capture tabular features, and we input them into the neural models. Inputting numerical sketches bypasses feeding table cell values to the model, addressing the token limit issue, as well as the numerical representation of cell values.

Following a phase of pretraining with sketches, we fine tuned the models for the data discovery tasks of union, join, and finding subsets. While some in the literature [15] have used self supervision for fine tuning, self supervision applies better for some tasks such as unions compared to say joinability. To finetune models for the data discovery tasks, we used a suite of 8 benchmarks across the 3 tasks *LakeBench* that we have open sourced for use [16]. On the fine tuning tasks, *TabSketchFM* outperformed other neural methods, by as much as 55% on F1. We then used the fine tuned models for search tasks, and showed that *TabSketchFM* outperforms the state-of-the-art neural and traditional methods by as much as 70% on F1.

Importantly, we also compare the performance of specialized traditional and neural models for data discovery against

[1]https://gdpr-info.eu

an off-the-shelf pretrained model that encodes column values as a single sentence. To our surprise, the off-the-shelf model performs much better than existing systems - in fact, for certain tasks such as union, it is sufficient to use that model for discovering unionable tables; our tabular specific model does worse, as do all other models that target unionability. The insight from this finding is that while some data discovery tasks need values to match across columns (e.g. join or subsets), for union, it is sufficient when columns have the same semantic meaning; indeed there is no need for any value overlap. Sentence embeddings capture those semantics well. Because certain tasks seem to benefit from these sentence embeddings (e.g. union), and other do not (e.g., subset), we propose a way to concatenate the embeddings from sentence encoders with tabular model embeddings, and show that one can augment tabular embeddings with value embeddings to improve performance, at least in some cases.

In summary, our contributions are as follows:

- **A novel tabular pretrained model.** We are the first to present a sketch-based tabular pretrained model, *TabSketchFM*, which contextualizes columns based on their data sketches and tabular metadata. The sketches help to better handle input token limits and numerical cell values. To achieve this, we modified the text-based transformer architecture to take in numerical and MinHash sketches as input. A purely sketch based approach though misses the opportunity to include table semantics from column values. Given that the inputs to the transformer architecture in our system are numeric vectors, it is difficult to combine these numeric inputs with tokens derived from cell values. We therefore combine the sketch based model embeddings with off the shelf embeddings of column values from existing pretrained sentence transformer models. We show that this combination boosts performance on some data discovery tasks.
- **Finetuning models for downstream tasks.** We develop cross-encoders from pretrained TabSketchFM, which we fine-tune for a range of downstream tasks from the LakeBench collection [16]. We fine tune not only *TabSketchFM* but other tabular pretrained models to see if they can perform well on these downstream tasks and present the comparision.
- **Detailed ablation study.** We assess the effectiveness of different sketches for identifying different types of table relevance. Our study presents interesting insights such as Minhash sketches of column values are crucial for identifying joins, but numerical sketches are crucial for identifying subsets; once again highlighting the value of using sketches for table representation.
- **Demonstration of TabSketchFM on search.** We compared the performance of *TabSketchFM*'s fine tuned models against neural and non-neural baselines. In the case of join search, *TabSketchFM* outperformed the nearest systems geared for join search by almost 70% in F1 scores, but tellingly, the sentence encoding model outperformed existing systems as well by 63%. By combining the two embeddings (from TabSketchFM and sentence encodings), we gain almost 73% over other systems in F1.
- **Illustrate generalization.** Finally, we illustrate the main advantage of learning a pretrained foundational model by showing the generalization of search performance across tasks. For example, we fine tune the pretrained model on join identification on one dataset and use it for union search on another dataset. Our evaluation shows that our finetuned models generalize well.

II. Related Work

Tabular Pretrained Models. Neural models are pretrained on tabular datasets to either recover masked tokens in the table or to detect corrupt values [8]–[11]. [17] surveys neural models for tabular data representation and their applications. [8] combined the Masked Language Model (MLM) objective from BERT [7] with a novel Masked Entity Recovery to train TURL. [10] use MLM with novel Cell-level Cloze and Table Context Retrival for TUTA. [11] used Masked Column Prediction and Cell Value Recovery for TABERT while [9] repurposed ELECTRA's objective function [18] for TABBIE. These models are finetuned and evaluated for different downstream tasks such as table metadata prediction, table content population, fact-checking, question answering, semantic parsing, and so on. For example, TABERT is evaluated on neural semantic parsing; TABBIE for column, row, and column type population tasks; and TUTA is finetuned and evaluated for cell and table type classification tasks. These downstream tasks take either a table or a table with a question as input; they do not consider the inter-table tasks such as dataset discovery, which we consider in this paper. Furthermore, they adopt serialization techniques from LMs and represent table values as sentences, whereas we represent the tables efficiently by using sketches.

Dataset Discovery Methods. Different methods such as keyword search [19], [20] and table search [21]–[23] have been used for the dataset discovery over data lakes. Table Union Search (TUS) [4] presented finding top-k data lake tables that are unionable with a query table. They determine the column unionability using three statistical tests based on value overlap, semantic overlap, and word embedding similarity, and aggregate them to infer table unionability. D^3L [24], adopted value overlap and word embedding measures from TUS and added three additional measures to measure column unionability: numerical column distributions, column header similarity, and regular expression matching. SANTOS [6] considers the relationships between the column pairs in both query and data lake tables. [15] developed a contrastive-learning-based table union search technique called Starmie that captures the context of the whole table in the form of column embeddings. AutoTUS [25] searches for unionable tables by contextualizing entire tables with embeddings of the relationship between the column pairs. Furthermore, there are other table search techniques that focus on finding joinable tables, i.e., given a query table marked with a query column, find the data lake

tables that contain a column with overlapping values with the query column. For instance, LSH Ensemble [13] defines joinable table search as a problem of finding data lake tables that have a column having high set containment with the query column. They propose a novel index that approximates set containment computation. JOSIE [5] searches for top-k joinable tables based on exact set containment. PEXESO [26] efficiently searches for the top-k semantically joinable tables. Along with a join on exact values, they also consider fuzzy join, i.e., join on abbreviations and synonyms. DeepJoin [27] is a deep-learning-based technique that also searches for semantically joinable tables. WarpGate [28] is also a system for searching for top-k semantically joinable tables by embedding each column to vector space using pre-trained embeddings. They index column embeddings using Locally Sensitive Hashing [29] for efficient search. Recently, Chorus [30] even demonstrated the ability of Large Language Models to identify joinable columns as a task of completing 'pd.merge' command when two tables are presented as pandas dataframe. Our work addresses the task of discovering tables from the data lakes that can be unionable, joinable, and/or subsets of each other. Notably, we specialize data discovery neural models based on a single pretrained model. The advantage of our approach over above mentioned task specific approaches is that, as in natural language processing, much smaller datasets are sufficient to finetune the model for new data discovery tasks; by leveraging the existing tabular knowledge in the pretrained model.

Benchmarks. Several tabular benchmarks exist in the literature [17], [31]. Koutras et al. [32] created benchmarks for schema matching [33] using open data tables and evaluated the existing matching techniques over them. Mudgal et al [34] open-sourced 13 datasets such as the DBLP-Google Scholar author dataset and Walmart-Amazon product datasets for entity matching task [35]. The WDC Web Table corpus [36], containing around 233 million web tables extracted from different web pages, has been widely used for diverse tasks such as Question Answering [12], [37], Semantic Parsing [11], [37], [38], Table Retrieval [39], Table Metadata Prediction [8], [10], [40] and more [41]. Efthymiou et al. [42] used knowledge graphs to create tabular benchmarks for column type prediction and column-to-column binary relationship prediction tasks. The VizNet Benchmark [43] has been used to evaluate column type prediction tasks [44], [45]. GitTables [31] is a large table corpus in the literature. However, they contain web tables that are small entity tables with fewer rows and columns than enterprise tables and do not represent enterprise data lake scenarios [5]. But the most closely related benchmark to our work is the LakeBench collection of datasets [16] for identifying relation between pair of tables (Sec. V) Hence, we use that benchmark for finetuning our model. Further, there are publicly available table search benchmarks that are also relevant to our work which we use for illustrating the use of TabSketchFM for data discovery. TUS [4] and SANTOS [6] are table union search benchmarks which we include in our experiments. At the time of writing this paper, there were no search benchmarks for table join search and table subset search, so we generated *Wiki join search* and *Eurostat subset search* benchmarks (Sec. VI-C). However, recently two new benchmarks have appeared in this space. One is manually annotated table join and union search benchmark [46] (also called LakeBench[2]) and another is a semantic table search [47] benchmark that used wikipedia categories and navigational links to establish semantic join and union relation.

III. TabSketchFM

In this section, we detail the architecture of TabSketchFM. Like existing tabular models [10], [12], we pretrain a transformer-based BERT model [7], but change the architecture to accommodate numerical inputs. The goal of building the pretrained model is to create cross encoders that can be used for ranking in data discovery using significantly smaller sized labeled datasets.

The first significant departure from other tabular representation models is in how we represent the table as input. While other models linearize table cells as text, we create *a sketch for each column* which are numerical vectors that need to be translated into a form that can be consumed by the transformer architecture. The sketches capture different features of the columns. Figure 1 shows an overview of how the different sketches we describe below are fed to the self-attention layers. Specifically, we create the following sketches for each column.

1. Numerical sketches. For each column, we record the number of NaNs and the number of unique values (normalized by the number of rows in the table). For string columns, we also record cell width in bytes, since that can be an important determinant in governing if the column is likely to be a join column. For instance, very long strings are unlikely to be join candidates. When possible, we convert date columns into timestamps and then treat them as numeric columns. For numeric columns, we additionally compute a percentile sketch, and add the mean, the standard deviation, and minimum and maximum values. All these features are encoded as individual elements of a single input vector we call the numerical sketch.

2. MinHash sketches. For each column, we compute a MinHash signature of cell values.[3] As shown in Fig. 1, the cell value of `Austria Vienna` is used as a single element in the set passed to the MinHash computation.[4] For string columns, we also compute another MinHash signature for the tokens within the column. The rationale for this second MinHash is that the tokens can sometimes capture its semantics (e.g., if a token `street` appears in two different columns, they maybe both about address, which is useful if one considers a union of two tables that do not necessarily share cell values). As shown in Fig. 1, the cell values of `Austria` and `Salzburg` get tokenized and used as two different elements of the set passed to the MinHash computation. We concatenate both MinHash

[2]Not to confuse with Srinivas et al. [16] LakeBench that we use for finetuning.

[3]In our implementation, we use datasketch library [48] to compute MinHash.

[4]While it does not make sense to convert float or integer columns into MinHash signatures, we do so because it is often difficult to tell if a column is truly a float or an integer, or it really is a categorical value.

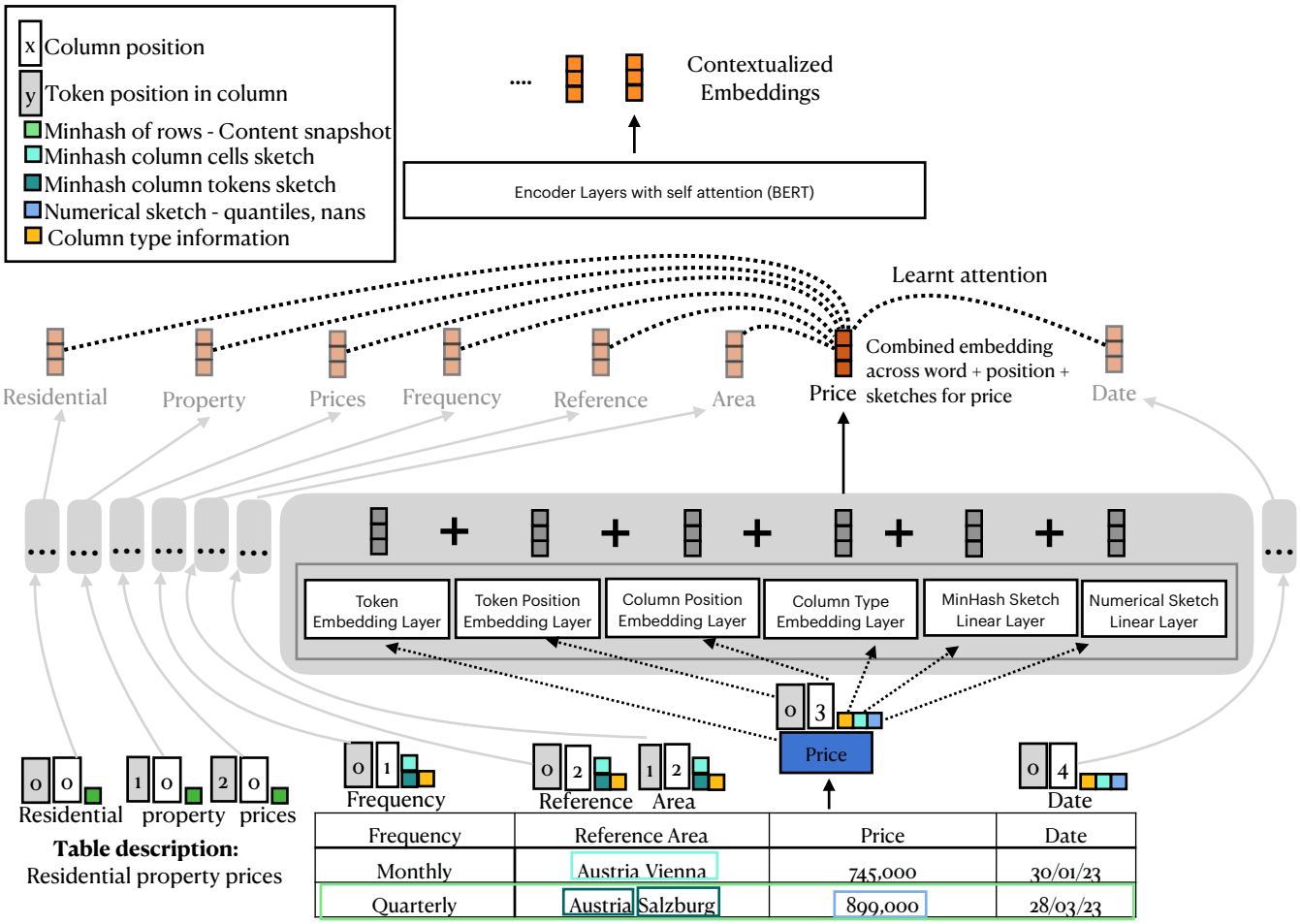


Fig. 1: Incorporation of sketches into transformer architecture. MinHashes are computed over the whole cell value, and tokens of a cell value for a given column. MinHashes are also computed over the entire row, treated as a string. For a given column token (e.g., Price in this figure), hidden states for each encoding are summed after passing through an embedding or a linear layer before it gets sent to the BERT encoder layers. Table description tokens are summed with MinHashes over the entire row.

sketches into a single input vector for string columns. For numerical and date columns, only the MinHash for the cell values is included in the input vector.

3. Content snapshot. Row-based information could be crucial in understanding the table semantics for data discovery [6]. We concatenate values within each row to form a string and create MinHash signature for each of them. For instance, we concatenate the last row of table in Fig. 1, surrounded by a green rectangle, into "*Quarterly Austria Salzburg 800,000 28/03/23*" and hash it as another minhash style sketch.

Now that we have these sketches, which are arrays of numbers, a key question is how to adapt BERT's architecture to consume these numeric values. Transformer models take sentences as input, and internally break it into different inputs such as the tokens of a sentence[5], the position of tokens in the sentence, and a scalar type indicating if the token belongs to the first sentence or the second. We leverage this mechanism to pass the table-specific scalar inputs as shown in Fig. 1 to the BERT Embedding Layers. For vector inputs (e.g., numerical sketch, MinHash sketch), we expand the notion of BERT Embeddings to add some linear layers, as described below.

TabSketchFM's specialized layers for processing tabular input are shown in the light gray box, middle-right of Fig. 1. The token `Price` for the `Price` column is used as an example for illustrative purposes. All other tokens belonging to the table description and other columns would undergo a similar process to create a single embedding per token.

1. Token Embedding Layer. The embedding layer of the BERT uncased model [7] contains numerical embeddings for each word in its vocabulary based on extensive training with a large corpus, such that, for instance, the embedding of `Price` might be closely associated with that of `House` because they co-occur often in text. Our model's token embedding layer is initialized with the weights of the BERT model's embedding

[5]The BERT tokenizer sometimes breaks a single word into multiple tokens such that unknown words in its vocabulary are handled correctly.

layer to leverage such information present in natural language, but the weights are allowed to be changed further while pretraining to include co-occurrence information in table-specific corpora. As shown in Fig. 1, the token `Price` would be mapped from its position in the vocabulary of the model (also pre-populated with BERT's vocabulary) into its hidden state using weights from this token embedding layer.

2. Token Position Embedding Layer. Each column is analogous to a sentence in a paragraph. Hence, we re-purpose token positional embeddings to reflect a token's position within a column name. For instance, as shown by the light gray scalar input in Fig. 1, the token position of a token `Area` is 1 because it is the second token in the cell (after `Reference`).

3. Column Position Embedding Layer. Columns themselves have positions and are encoded through a new embedding layer which can range from 1 to the total number of columns,[6] as shown by the white scalars in Fig. 1. The rationale for including column positions is that they sometimes do have semantics; e.g., a street address next to the city, followed by a zipcode. Of course table semantics do not change as a function of column order; nevertheless, we included position in case it helps the model understand column semantics, with the assumption that the attentional mechanism would disregard order if necessary.

4. Column Type Embedding Layer. Column types of string, date, integer, or float are encoded through another embedding. In order to determine column type, we made a best-case effort to parse the first 10 values of each column as dates, integers, or floats and defaulted to string if we could not convert them. In mixed-type columns, this can yield poor results, but at the very worst, at least one of the types was assigned to these columns. Column types are shown in Fig. 1 in orange.

5. MinHash Sketch Linear Layer. MinHash sketches and the content snapshots were passed through a linear layer to encode them into the same hidden state dimensions of the existing transformer layers in Bert.

6. Numerical Sketch Linear Layer. Numerical sketches are similarly passed through a linear layer to encode them.

As shown in Fig. 1, for each column token, we combined the hidden states of its token embedding, token position embedding, column position embedding, column type embedding, MinHash sketch embedding, and numerical sketch embedding using summation. This summation is similar to the summation of token embeddings and position embeddings in the original BERT model. Content snapshot signatures are summed with table description tokens because the content snapshot describes the entire table's content. For instance, for the tokens `Residential`, `Property` and `Prices`, the content snapshot is passed into the linear layers to create a hidden state that is combined with the other embeddings for those tokens. Once each token has been summed with its other sketches, positional and type encoding, it is passed to the rest of the encoder and attention layers from BERT. The attention layers are crucial for contextualizing the embeddings based on what other columns, values, and table descriptions exist in the table. As shown in the figure for `Price`, the attention mechanism learns N^2 attention weights between all N tokens for a given table. This step will consider `Price` in relation to all other columns as well as table descriptions. We used the usual 12-layer BERT encoder model with self attention from HuggingFace[7] once we have unified the hidden states from the embedding layers and the linear layers.

IV. Pretraining

Dataset. Existing works generally use non-public data [17] or web tables [36] to pretrain the models. Unlike enterprise data, web tables often have few rows and columns to ease human consumption [19] and they focus on entities popular on the web (e.g., countries). Their performance may not generalize to enterprise data, where the tables have a large number of rows, the entities are often domain-specific, contain cryptic code words, and have a lot of numerical information. Therefore, we created a de-duplicated pretraining dataset of $197,254$ enterprise-like tables (CSVs) from CKAN and Socrata that are legally cleared to have the open licenses. This pretrain dataset contains 2234.5 rows and 35.8 columns in each table on average. About 66% of columns were non-string, resembling an enterprise datalake.

Data Augmentation. In the image processing literature [49], different transformations are applied over an original image to generate new image samples for training. This makes the trained model robust on different variants of the same image. We want the model to be robust to the order of columns such that shuffling columns in a table does not impact its semantics. So, we created three different versions of the table, by changing the column order, which in turn, changed the table's content snapshot (see Section III). This increases the total number of pretraining tables to 290,948 and for each of them, we create signatures as explained in Section III.

Method. We employ the standard Mask Language Modeling (MLM) as the pretraining objective for our model, as in most text-based language models. First, a few tokens are randomly sampled according to the masked language probability. Then they are substituted with a [MASK] token to create an input sequence. The input sequence is fed to the model, and the model is asked to predict the substituted words. This can be viewed as a classification task where the original word is the label and the model's entire vocabulary is the set of possible labels. Cross-entropy loss is computed for each predicted word in the input text that was masked.

$$\frac{-1}{N}(\sum_{n=1}^{N} y_i * log(\hat{y_i})) \tag{1}$$

Equation (1) defines cross-entropy loss, where y_i is the class labels in the ground truth, $\hat{y_i}$ is the predicted class label and N is the number of training examples in the train set.

[6]Position 0 is reserved for any tokens in the table description.

[7]https://huggingface.co/docs/transformers/model_doc/bert

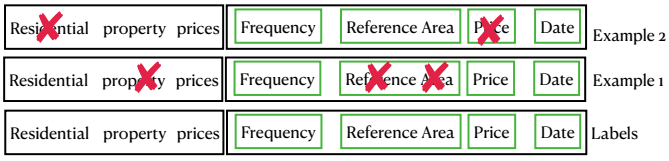


Fig. 2: A combination of whole column masking, and MLM probability masking to generate up to 5 examples. The box on the left hand side is the table description and the one on the right hand side is the column names

In our scenario, for each table, we mask a single column name such that all tokens corresponding to that column are masked. This is analogous to the natural language literature's whole word masking where all tokens corresponding to a word are masked. In addition, we used MLM probability to mask the tokens in the table description as well. As shown in Fig. 2, we generate several samples from a single table. Note that each masking gives us an example. For small tables with less than 5 columns, we mask each of its columns one after another. However, if a table has a large number of columns, this produces a lot of examples for the same table, over-representing it. So, for large tables with more than 5 columns, we randomly select five columns and mask them. Following this strategy over the augmented pretraining dataset, we get $730,553$ examples in training, $54,430$ examples in validation, and $58,141$ examples in test sets. We provide the experimental setup for pretraining in Section VI.

V. Fine-tuning

To use our neural tabular model for identifying relevant tables in the data lakes, we finetune TabSketchFM on the *LakeBench* collection [16]. LakeBench is a collection of 8 benchmark that includes data from various sources such as open government data from CKAN and Socrata, economic data from the European Central Bank [50], Spider [51], and synthesized data from large knowledge graphs such as Wikidata [52]. The benchmarks cover binary classification, regression, and multi-label classification for the tasks of identifying table unionability, table joinability, and table subset. It contains 3 datasets for identifying unionable tables—TUS-SANTOS, Wiki Union, and ECB Union; 4 datasets for identifying joinable tables—Wiki Jaccard, Wiki Containment, Spider-OpenData, and ECB Join; and one dataset for identifying table subsets—CKAN Subset. Of these datasets Wiki Jaccard, Wiki Containment and ECB Union are formulated as regression task; ECB Join is multi-class classification; and remaining are classification tasks. Table I summarizes the main characteristics of the datasets to show the type of tasks we used for fine tuning.

For each finetuning task, we create a cross-encoder [53], which evaluates if a pair of tables can be joined, unioned or are subsets of each other. We use cross-entropy loss for classification tasks, mean squared error for regression tasks, and binary cross-entropy with logit loss for multi-class classification tasks. The goal of building multiple cross encoders for the same task across different datasets was to examine whether the cross encoders would generalize across datasets; allowing re-use to new data with no additional training.

VI. Experiments

Our empirical work is organized around the following research questions:

Q1. How does sketch-based TabSketchFM compare to existing pretrained tabular models when finetuned for relevant table identification tasks in LakeBench?

Q2. How do different sketches of TabSketchFM impact different tasks?

Q3. How well do these fine tuned TabSketchFM models perform on data discovery tasks?

Q4. Can TabSketchFM fine tuned on a single task and domain, adapt and generalize to a new domain?

We pretrain TabSketchFM using 4 A100 40GB GPUs for 2 days, until the model converged. Our pretrained model contains $118M$ parameters, similar to other row-based models in our experiments. We use patience of 5, i.e., we consider the model as having converged if the validation loss does not decrease for more than 5 epochs. As studied in the literature [54] this not only reduces training time but also helps us avoid overfitting. For finetuning the cross encoders, for all tasks, we use one A100 40GB GPU. Most finetuning tasks converged within 6 hours using the same 5 epoch patience as pretraining except for the Wiki Union benchmark which took 24 hours. Our code is accessible at https://anonymous.4open.science/r/tabsketchfm.

A. Fine-tuning models on LakeBench Tasks

We evaluate our finetuned cross-encoders and the existing value-based tabular foundational models for the LakeBench unionability, joinability, and subset tasks.

1) Baselines: We compare TabSketchFM against *four* publicly available tabular foundational models: TUTA, TaBERT, TABBIE, and TAPAS. We adapted these models for Lakebench data discovery tasks by building a dual encoder architecture. Each encoder represents the pretrained model with shared parameters; and the same model is used to encode each element of a table pair. The embeddings from the last layer of the encoders were concatenated and passed through a two-layered MLP (multi-layered perceptron). We used this architecture as opposed to the cross-encoder architecture we used for TabSketchFM because in general it is unclear how to feed two different tables with different row sizes into a cross-encoder. It gets even more complicated when one considers models such as TUTA that model hierarchical information in the table as a tree. While we were able to finetune the TUTA and TaBERT code in this dual encoder architecture, the code of TAPAS and TABBIE were less amenable to finetuning. Hence, for TAPAS and TABBIE, we froze their pretrained models while finetuning, but allowed the two layers above the model to learn the weights. In this case, the pretrained model can be seen as producing an embedding that is then fed into two layered network that tunes for a specific task. We will make

TABLE I: Cardinality of all the datasets in LakeBench, as well as search benchmarks in this paper.

Benchmark	Task	# Tables	Avg. Rows	Avg. Cols	# Table Pairs			Data type distribution (%)			
					Train	Test	Valid	String	Int.	Float	Date
TUS-SANTOS	Binary Classification	1127	4285.17	13.04	16592	3566	3552	77.94	8.62	7.51	5.93
Wiki Union	Binary Classification	40752	51.05	2.66	301108	37638	37638	57.97	14.38	24.25	3.4
ECB Union	Regression	4226	292.47	36.3	15344	1910	1906	47.72	14.05	36.31	1.92
Wiki Jaccard	Regression	8489	47.3	2.8	12703	1412	1572	57.5	15.66	19.76	7.07
Wiki Containment	Regression	10318	47.15	2.79	21007	2343	2593	57.26	15.26	20.58	6.9
Spider-OpenData	Binary Classification	10730	1208.87	9.09	5146	742	1474	42.22	18.51	32.62	6.66
ECB Join	Multi-label Clasification	74	8772.24	34.47	1780	222	223	52.14	7.8	37.79	2.27
CKAN Subset	Binary Classification	36545	1832.58	25.37	24562	2993	3010	31.75	17.53	46.14	4.58
Eurostat Subset	Search	38904	2157	10.46				64.63	9.03	7.83	18.50
Wikijoin	Search	46521	49.64	2.68				58.13	13.35	25.0	3.50

TABLE II: Performance (avg $\pm$ stdev) of TabSketchFM vs. other baseline models on LakeBench averaged across five random seeds. **Best** performance is highlighted in bold, and the second best is underlined.

Tasks	Vanilla BERT	TAPAS	TABBIE	TUTA	TaBERT	TabSketchFM (ours)
TUS-SANTOS (F1)	$\mathbf{0.99 \pm 0.00}$	0.34 ± 0.00	$\underline{0.75 \pm 0.22}$	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.99 \pm 0.00}$
Wiki Union (F1)	0.33 ± 0.00	0.41 ± 0.00	0.64 ± 0.12	0.33 ± 0.00	$\mathbf{0.97 \pm 0.00}$	$\underline{0.94 \pm 0.00}$
ECB Union (R2)	0.03 ± 0.03	-0.01 ± 0.01	0.02 ± 0.01	$\underline{0.87 \pm 0.01}$	0.35 ± 0.06	$\mathbf{0.90 \pm 0.03}$
Wiki Jaccard (R2)	0.00 ± 0.00	-0.03 ± 0.03	0.25 ± 0.02	$\underline{0.43 \pm 0.02}$	0.33 ± 0.03	$\mathbf{0.58 \pm 0.03}$
Wiki Containment (R2)	0.00 ± 0.00	0.00 ± 0.00	0.21 ± 0.01	$\underline{0.35 \pm 0.03}$	0.30 ± 0.03	$\mathbf{0.58 \pm 0.02}$
Spider-OpenData (F1)	0.71 ± 0.06	0.65 ± 0.00	0.57 ± 0.00	0.76 ± 0.12	$\mathbf{0.87 \pm 0.02}$	$\underline{0.83 \pm 0.01}$
ECB Join (F1)	0.63 ± 0.04	0.40 ± 0.00	0.42 ± 0.03	$\underline{0.81 \pm 0.01}$	0.79 ± 0.03	$\mathbf{0.86 \pm 0.01}$
CKAN Subset (F1)	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	$\underline{0.43 \pm 0.00}$	$\mathbf{0.98 \pm 0.00}$

the code for all the baselines available. Below, we provide details on model-specific implementation issues.

- **TaBERT** provides two types of embeddings for a table: context embeddings corresponding to the tokens in the context, and column embeddings corresponding to each column in the table [11]. For each table, we compute both embeddings for top $10,000$ rows. In [11], a content snapshot approach is described which creates synthetic rows based on the n-gram matches of the input question with the cell values. We could not experiment with this feature, since 1. this technique is not part of the authors' provided source code[8] and 2. LakeBench tasks do not contain a question. So, we compute these embeddings for each table, mean-pooled over the context tokens and tabular columns respectively. The final embedding for the table pair is computed as the concatenation of context and column embeddings obtained for each table.
- **TUTA** creates a tree-based attention matrix over all the tokens in the table, which has significant memory requirements. To overcome this, we obtain the top 256 rows and columns of the table, and the first 256 tokens of the table sequence.
- **TAPAS** requires a natural language query and a table as input [12]. For our dataset discovery tasks, we sent an empty string as a natural language query and use 512 serialized row based version of the table as the embedding.
- **TABBIE** provides row and column embeddings of the table [9]. For our work, we obtained the row embeddings for each row in the table. Following the original work, we use the first 30 rows and 20 columns of the table. These row embeddings are combined using the mean operation and the resulting vector is frozen as table embedding.

We also added a naive baseline, **Vanilla BERT**, to examine to what extent LakeBench tasks can be performed based on column headers alone; this provides a measure of the difficulty of a given task. For *Vanilla BERT*, column headers were provided for the two tables as two sentences, and the BERT model was finetuned on the specific LakeBench task as were other models.

2) Results: Table II compares the performance of TabSketchFM with baseline models on LakeBench. For regression tasks, we report R2 statistics, and for (binary and multiclass) classification tasks, we report a weighted F1 score to handle skew in classes[9]. The best performance is highlighted in bold and the second best is underlined. All the results presented are aggregated over 5 random seeds. We see that on *five of the eight tasks, TabSketchFM outperformed all baselines, performed as well as the best methods on one, and performed second best on the remaining two*.

[8]https://github.com/facebookresearch/TaBERT

[9]Our metrics were implemented from `scikit-learn`: https://scikit-learn.org/stable/modules/model_evaluation.html

The binary classification adaptation of the TUS-SANTOS benchmark, was too easy and could be solved on the basis of column headers alone[10], as evident from the performance of the Vanilla BERT model. Nevertheless, TAPAS and TABBIE perform relatively poorly on it, pointing to the importance of including the Vanilla BERT as a baseline.

On the CKAN Subset benchmark, we find that the performance of most models was comparable to random guessing since the column headers were exactly the same, and most systems (except TaBERT) did not have a view of the entire dataset. On Wiki Union and on Spider-OpenData, TaBERT outperformed TabSketchFM. Even so, TabSketchFM achieved the second best performance on these two tasks.

We also performed an error analysis on table pairs that TabSketchFM got wrong on Wiki-Union, Spider open data and CKAN. In all these cases, mean hamming distances of MinHash sketches for positive pairs were equivalent to those from negative pairs; suggesting that the sketches alone did not contain sufficient information to discriminate those examples. TaBERT's superior performance on the Wiki Union case, combined with the fact that lack of discrimination of MinHashes provides a clue to at least one weakness in the current version of TabSketchFM, which is that we do not consider any values from the column at all. As shown in Figure 4, two columns may be unionable when their values map to the same domain (e.g., different municipalities in Slovakia), with minimal overlap in actual values. In such cases, a system such as TaBERT can still determine that the cell values are similar, and belong to the same semantic type, whereas TabSketchFM cannot. We show how to incorporate cell values externally in the section on search which addresses this issue.

In summary, the answer to Q1 is that sketches-based TabSketchFM is in general superior to other pre-trained models for data discovery in LakeBench. Note that this evaluation does not directly speak to performance on search tasks in data discovery; but it does evaluate the usefulness of sketches for building a good model tuned for data discovery. We turn next to investigating how different sketch types help specific tasks, before returning to the question of how well these fine tuned models perform on search.

B. Effectiveness of sketch types on task types

To explore the relevance of the sketches and content snapshots for the different data discovery tasks, we conduct an ablation study where we use only one type of sketch. The results on LakeBench tasks with the random seed 0 are shown in Table III. In Table IV we present another ablation study where we remove one sketch and retained the rest. Not surprisingly, MinHash sketches are crucial for join tasks, where MinHash alone seems to dominate overall performance of *TabSketchFM*. Subset selection clearly relies on the overlap of the data distribution; numerical sketches alone match the performance of *TabSketchFM*. Given that almost 69% of the columns were numerical in the subset benchmark, this outcome might be expected. The effect of content snapshot was less clear. Looking at Table IV, it is evident that the removal of content snapshot does in fact contribute somewhat to the Wiki join tasks. Numerical sketches contribute to ECB union which is perhaps because approximately 50% of the columns are not strings. Removal of MinHashes mirrors the result from earlier, which is, join tasks are most affected. Surprisingly, the removal of MinHashes affected the subset task; even though numerical sketches by themselves produced good performance. It is possible that there are complex interactions between the sketches. **In summary,** we answer Q2 from our ablation studies that different sketches play different roles across tasks, and may even interact to determining data discovery performance.

C. Application to search

We evaluate the embeddings from our finetuned TabSketchFM for three data discovery tasks: join search, union search, and subset search.

1) Join Search: Here, the task is to find all the tables from the data lake that can be joined on one or more columns with the given query table. Note that prior join works [5], [26], [55] evaluate the efficiency of searching for data lake columns having value overlap with the query column(s), and assume that the columns with overlapping values are always joinable. However, we are interested in whether it makes sense to join tables. For example, two columns: (i) explaining people's Ages with values such as (50, 24, 66, 78, 92, 95) and (ii) explaining students' marks out of 100 (rounded to integer) with values such as (78, 92, 95, 100, 20) can have overlapping values but it is not sensible to join them because they have different semantics. Because TabSketchFM is built to generate contextual embeddings of columns that include value overlap, column headings, its context with other columns, and so on, it should be possible to get rid of such useless joins.

To our knowledge, there are no such benchmarks with annotated gold standards that consider if it is sensible to join[11]. So, we construct a search benchmark named **Wiki Join** from Wikidata (by following the cell-entity mapping approach used in the curation of the Wiki Jaccard dataset [16], see Table I for dataset characteristics). We first generated the tables and then used the ground truth cell-entity mapping to create a list of all the pairs of columns with the same entity annotations. For example, consider Figure 4. All entities in first column of table A and first column of table B are mapped to `Meteorite` where as entities in table C are mapped to `Locations`. So even when there is an overlap in cell values of all tables (`Aleppo` appears in all the tables), only table A and B are considered sensible to join. Once the list of sensible joinable column pairs is obtained, we assign each pair an overlap score, which is the Jaccard score of the entity annotation sets. The pairs with overlap score greater than 0.5 were considered as

[10]We added column headers for this benchmark, unlike in the literature, but removing headers made little difference to the results of tabular models.

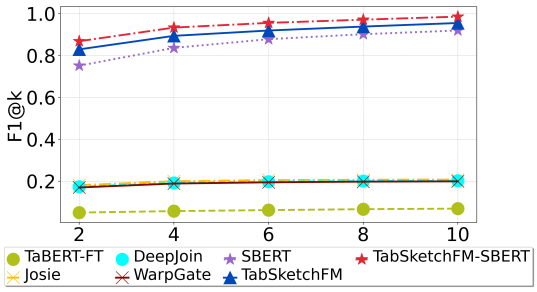
[11]A new benchmark very recently got published on this topic but that was after we had built the benchmark [46]

TABLE III: Importance of different sketches on *TabSketchFM* performance for the different tasks. Performance of sketches that are equivalent (within 2 points) to overall performance is highlighted in green. TUS-SANTOS is not considered because it can be performed based on column headers alone.

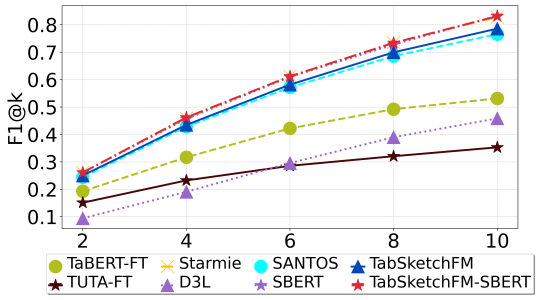
Tasks	Using only MinHash sketches	Using only Numerical sketches	Using only Content snapshot	*TabSketchFM* (w/ everything)
Wiki Union (F1)	0.914	0.804	0.897	0.940
ECB Union (R2)	0.829	0.498	0.752	0.897
Wiki Jacc. (R2)	0.537	0.318	0.314	0.577
Wiki Cont. (R2)	0.628	0.252	0.301	0.587
Spider-Op. (F1)	0.831	0.817	0.797	0.831
ECB Join (F1)	0.874	0.812	0.815	0.856
CKAN Subset (F1)	0.431	0.984	0.431	0.986

TABLE IV: Removing a sketch to study the impact of it on *TabSketchFM* performance for the different tasks. Performance of sketches that declined more than 2 points compared to overall performance is highlighted in red.

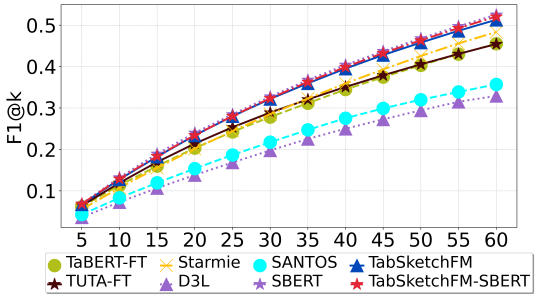
Tasks	Removing only MinHash sketches	Removing only Numerical sketches	Removing only Content snapshot	*TabSketchFM* (w/ everything)
Wiki Union (F1)	0.933	0.927	0.931	0.940
ECB Union (R2)	0.770	0.872	0.897	0.897
Wiki Jacc. (R2)	0.425	0.565	0.519	0.577
Wiki Cont. (R2)	0.358	0.598	0.559	0.586
Spider-Op. (F1)	0.814	0.851	0.847	0.831
ECB Join (F1)	0.812	0.855	0.846	0.855
CKAN Subset (F1)	0.431	0.431	0.980	0.986



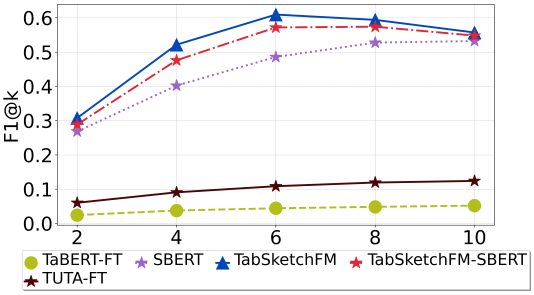
(a) F1 on *wiki join* search for varying *k*.



(b) F1 on *Santos* search for varying *k*.



(c) F1 on *TUS* search for varying *k*.



(d) F1 on *Eurostat* subset search for varying *k*.

Fig. 3: F1 plot for search on various join, union and subset datasets.

joinable in our dataset for a total of 5,512 queries. This dataset contains contains $46,521$ tables and $5,512$ query columns.

We compare the join search performance of TabSketchFM with two traditional join search methods—**LSHForest** and **Josie** [5]—and three neural methods–**WarpGate** [28], **DeepJoin** [27], and **TaBERT**. WarpGate is a recent embedding based method that performs joinable column search using SimHash LSH in the embedding space. It uses FastText embedding pretrained on WebTables to generate embeddings of column values. DeepJoin is a similar embedding base approach that finetunes a pretrained language model and uses HNSW [56] for indexing and efficient search. They experiment with FastText, BERT, DistilBERT and MPNet. In our experiments we use the FastText embedding as it was found to be the best performing embedding when used without finetuning. In addition to column values, DeepJoin also uses column names, table names and column statistics (max, min and average character length) while embedding; much like our sketches. For TaBERT baseline, we use the column embeddings obtained from the TaBERT model which was finetuned on the Wiki-Containment dataset from LakeBench in Section VI-A (**TaBERT-FT**). We could not include TUTA as it does not provide column embeddings.

Recognizing the power of pretrained models trained on massive corpora of text, we added an off-the-shelf sentence encoder **SBERT** (`all-MiniLM-L12-v2`) as a powerful baseline. We include a very simple approach of concatenating the top 100 unique values in a column into a single sentence

TABLE V: F1, Precision & Recall for Wiki-join search.

Baseline	Mean F1	P@10	R@10
TaBERT-FT	5.88	0.43	0.04
LSH-Forest	10.48	0.8	0.08
Josie	19.56	**0.99**	0.12
DeepJoin	18.88	0.96	0.11
WarpGate	18.58	0.95	0.11
SBERT	83.67	0.95	0.89
TabSketchFM	89.09	0.97	0.94
TabSketchFM-SBERT	**92.81**	0.98	**0.99**

TABLE VI: F1, Precision & Recall for SANTOS union search.

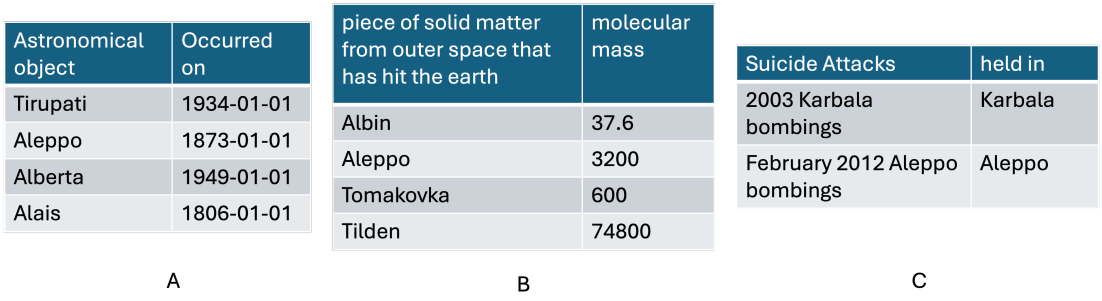
Baseline	Mean F1	P@10	R@10
TaBERT-FT	36.64	0.63	0.46
TUTA-FT	25.34	0.43	0.3
Starmie	**54.08**	**0.97**	0.72
D3L	26.44	0.54	0.4
SANTOS	50.36	0.89	0.67
SBERT	53.86	**0.97**	**0.73**
TabSketchFM	51.38	0.92	0.69
TabSketchFM-SBERT	**54.09**	**0.97**	**0.73**

TABLE VII: F1, Precision & Recall for TUS union search.

Baseline	Mean F1	P@60	R@60
TaBERT-FT	26.66	0.90	0.30
TUTA-FT	27.27	0.89	0.31
Starmie	27.48	0.96	0.32
D3L	18.98	0.75	0.21
SANTOS	20.83	0.81	0.23
SBERT	**31.13**	**0.99**	**0.36**
TabSketchFM	30.43	0.97	0.35
TabSketchFM-SBERT	30.72	**0.99**	0.35

TABLE VIII: F1, Precision & Recall for Eurostat subset search.

Baseline	Mean F1	P@10	R@10
TABERT-FT	4.03	0.05	0.05
TUTA-FT	9.82	0.13	0.12
SBERT	43.12	0.56	0.51
TabSketch	**49.96**	**0.59**	**0.53**
TabSketch-SBERT	47.54	0.58	0.52



Astronomical object	Occurred on
Tirupati	1934-01-01
Aleppo	1873-01-01
Alberta	1949-01-01
Alais	1806-01-01

A

piece of solid matter from outer space that has hit the earth	molecular mass
Albin	37.6
Aleppo	3200
Tomakovka	600
Tilden	74800

B

Suicide Attacks	held in
2003 Karbala bombings	Karbala
February 2012 Aleppo bombings	Aleppo

C

Fig. 4: Sample subsets of four tables from our Wikidata-based table union and join benchmark (top), along with ground truth labels and mappings (bottom)

and encoding it to produced a column embedding. As seen in figure 3a this simple baseline was quite effective compared to many traditional hand crafted systems for data discovery, as well as other neural systems. For finding joins (and subsets), value overlap is a requirement, and data sketches do a very efficient job of finding overlap in large sets, whereas encoding many thousands of values in a sentence is inefficient and likely ineffective. To examine if there is any gain in combining the embeddings of columns derived from TabSketchFM models along with value embeddings for each column using sentence encodings, we also concatenated the two embeddings after normalizing them so the means and variances of the two vectors were in the same scale (**TabSketchFM-SBERT**).

Figure 3a shows the significant advantage of TabSketchFM over all other baselines, and furthermore, adding column embeddings using SBERT improved performance for joins. Table V shows the mean F1 scores for all methods. In this and other benchmarks, the best performing system and the runner up were better than all other methods, using statistical significance tests. Table V presents the Mean F1, Precision @ 10, and Recall @10 values.

2) Union Search: We use two publicly available benchmark datasets: $\mathbf{TUS_{Small}}$ and $\mathbf{SANTOS_{Small}}$ [4], [6] because the larger versions of the dataset are not labeled. As a baseline, we compare against state-of-the-art table union search techniques: $\mathbf{D^3L}$ [24], **SANTOS** [6], and **Starmie** [15] as well as the **SBERT** baseline explained earlier. We could not compare against $AutoTUS$ [25], a recent union search system, because its code was not public at the time of writing this paper.

To determine unionability, table embeddings from TabSketchFM did not yield strong results; perhaps because the union benchmarks had been created by creating subsets of the columns in a table or rows. Following the state-of-the-art unionability approach Starmie [15], we used column embeddings of TabSketchFM to determine if two tables are unionable. But we used a simpler technique than the bipartite graph

$$
\begin{aligned}
\textsc{KnnSearch}\,(c,k) &\equiv (\langle c_i, d_i\rangle \dots)\ (k\times 3) \text{ nearest columns } c_i \text{ by distance } d_i \\
\textsc{ColumnNearTables}\,(c,k) &\equiv \bigcup_{t_i\in\{\textsc{Table}(c_i)\mid c_i\in\textsc{KnnSearch}(c,k)\}} \left\langle t_i, \min_{\langle c_i,d_i\rangle\in\textsc{KnnSearch}(c,k)\wedge\textsc{Table}(c_i)=t_i} d_i \right\rangle \\
\textsc{NearTables}(t) &\equiv \left\{ \langle t_i, \{\langle c_i, d_i\rangle\}\rangle \;\middle|\; \left\langle \langle c_i, d_i\rangle \in \exists_{c\in\textsc{ColumnNearestTables}(t,k)} \textsc{ColumnNearestTables}(c,k) \right\rangle \right\} \\
\textsc{Rank}_1\,(\langle t_i, \{\langle c_i, d_i\rangle\}\rangle) &\equiv |\{c_i \dots\}| \\
\textsc{Rank}_2\,(\langle t_i, \{\langle c_i, d_i\rangle\}\rangle) &\equiv \textstyle\sum d_i
\end{aligned}
$$

Fig. 5: Ranking of nearest tables

matching algorithm introduced by Starmie. Our approach is defined in Figure 5, and works as follows:

1) For each column c in a given table t, KNNSEARCH returns the nearest $k * 3$ *columns* in the dataset. Note that we try to get a lot more columns than k simply to increase the number of candidate columns we consider at this step because multiple columns from a single table might match a given column.
2) For a column c, COLUMNNEARESTTABLES returns each table near c and the distance of its closest column.
3) For a table t, NEARESTTABLES gathers the near tables for each column in t.
4) We then choose the best, first using $\textsc{Rank}_1$ to select the tables with the largest number of matching columns, and then, for ones where that is equal, $\textsc{Rank}_2$ chooses the closest, i.e. the smallest sum of column distances.

For each technique, we use default parameters suggested in the respective papers. For TUTA we had to use table embeddings because that was what the model made available. For both the TUTA and TaBERT, we use the model that was fine tuned on TUS-SANTOS dataset from LakeBench in Sec. VI-A(denoted by **TUTA-FT** and **TaBERT-FT**). The results for TabSketchFM compared to the baselines show that TabSketchFM matched the best system [15] on SANTOS, when we added embeddings from SBERT, see Figure 3b, and Table VI for a summary across methods. For TUS, value embeddings from the off-the-shelf SBERT model performed very well, as did TabSketchFM when enhanced with value embeddings; see Figure 3c and Table VII. This suggests for union, column values alone are sufficient to find good unionable results; table specific embeddings do not seem to add much. Note that value embeddings out performed all the other systems, pointing to the power of existing pretrained models in embedding tabular data.

3) Subset Search: To our knowledge, there are no benchmarks for subset search. To ensure that the models we built generalize to search, we constructed a dataset for subset search, from Eurostat [12] of 3,072 CSV files comprising the variety of data collected by the EU. These files serve as query tables for search. Data characteristics for this benchmark are provided in Table I. For each file, as shown in Figure 6, we created the following variants reflecting different types of subsets of the file, sampling 25%, 50% or 75% or all of the rows or columns in the table. When all rows or columns were kept, we shuffled either the rows or the columns (shown in purple). These last two variants were added to measure whether the embeddings capture table semantics, that is order invariance for rows and columns of a table. The 11 variants per file made up a search dataset of 38,904 files.

We follows the same procedure described in Figure 5 to find tables that were subsets of the query tables. Figure 3d and Table VIII shows the results of the subset search on the fine tuned model trained on ckan-subset. As we see in the next section on transfer, this model turned out to be the weakest TabSketchFM models for the subset search problem, likely reflecting the quality of the dataset used for training The best performing TabSketchFM model was trained on union, on the European Central Bank data at 0.55. Still, we include it for consistency. In addition, the results suggest that for subsets, adding value embeddings of columns from SBERT actually made performance slightly worse. This highlights the importance of being able to customize embeddings for different data discovery tasks - the ability to add value embeddings to tabular embeddings seems to be useful for such customization.

One additional question we asked is whether the TabSketchFM models exhibited column order invariance, and row order invariance. Unsurprisingly, changing row order made no difference to TabSketchFM, all 3072 queried tables returned the row shuffled variants in their nearest neighbors set. In contrast, changing row order did influence SBERT embeddings; only 2809/3072 (91%) row shuffled variants were returned. Column order can matter to TabSketchFM models, since the encoder is a sequence model. For the column shuffled variants, 3059/3072 appeared in the nearest neighbors set (99.5%). The SBERT model of course did not consider the larger column context, and was hence unaffected by the shuffling of columns.

Overall, from our evaluations of 4 data discovery datasets across traditional and neural approaches answers Q3, that embeddings from finetuned TabSketchFM can be used for data discovery tasks. As the embeddings are pretty small vectors the search in embedding space can be made extremely efficient; making it an ideal candidate for large data lakes.

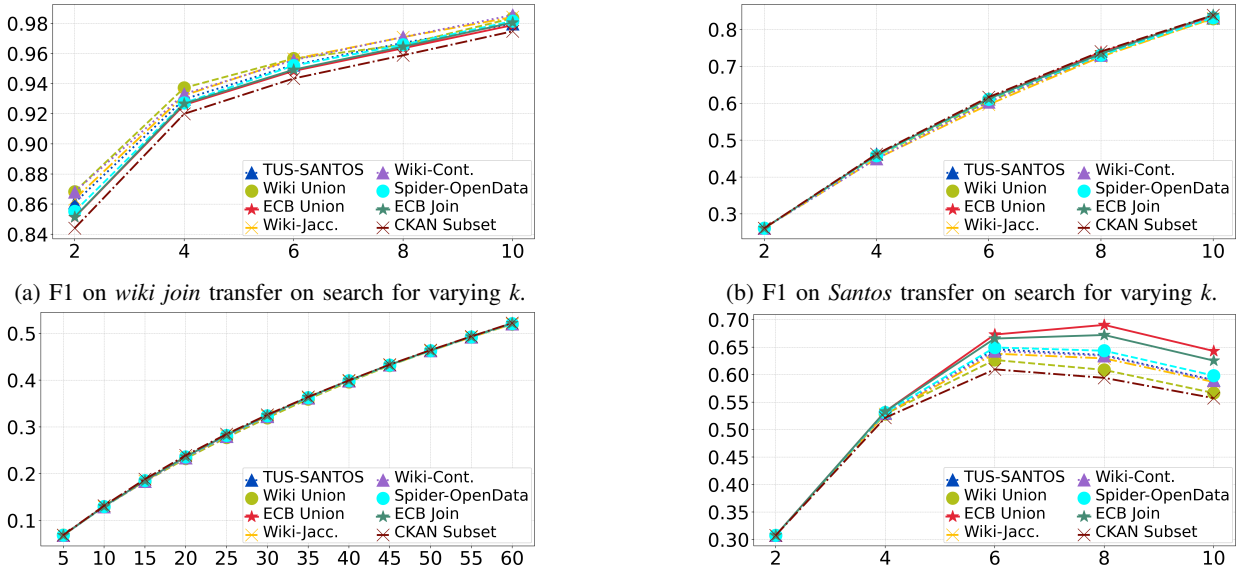
4) Generalization across tasks and domains: We now investigate Q4. How does our method, fine-tuned on a single task and domain, can adapt and generalize to a new domain.

[12]https://data.europa.eu/en

DATAFLOW	LAST UPDATE	freq	itm_newa	indic_ag	unit	geo	TIME_PERIOD	OBS_VALUE
ESTAT:AACT_EAA01(1.0)	15/05/24 11:00:00	A	1000	PROD_BP	MIO_EUR	AT	1990	1089.16
ESTAT:AACT_EAA01(1.0)	15/05/24 11:00:00	A	1000	PROD_BP	MIO_EUR	AT	1991	1007.4
ESTAT:AACT_EAA01(1.0)	15/05/24 11:00:00	A	1000	PROD_BP	MIO_EUR	AT	1992	850.53
ESTAT:AACT_EAA01(1.0)	15/05/24 11:00:00	A	1000	PROD_BP	MIO_EUR	AT	1993	859.92

25% rows 25% columns	50% rows, 50% columns	75% rows, 75% columns	100% rows, 25% columns	100% rows, 50% columns	100% rows, 75% columns	25% rows, 100% columns	50% rows, 100% columns	75% rows, 100% columns	100% rows, Shuffle columns	Shuffle rows, 100% columns

Fig. 6: Sample table and generated subsets from Eurostat



(a) F1 on *wiki join* transfer on search for varying *k*.

(b) F1 on *Santos* transfer on search for varying *k*.

(c) F1 on *TUS* transfer on search for varying *k*.

(d) F1 on *Eurostat* transfer on search for varying *k*.

Fig. 7: F1 plots for transfer across different K values. (a) WikiJoin (b) Santos (c) TUS (d) Eurostat

Figures 7a, 7b, and 7c, 7d, show that the fine tuned cross encoder models generalize rather well across tasks as well as domain, which is encouraging particularly because it shows that the models can be trained on a datalake that is quite different from the datalake that it is deployed to. In enterprise contexts, this is an important requirement, because it means that we can basically train models offline and apply it online. All models shown in the Figure are models that include the value embeddings from a column for maximal generalization, but using the models without values also produced the same type of generalization.

VII. Conclusion and Discussion

We presented TabSketchFM, a novel transformer-based tabular representation model that inputs the table in the form of several sketches. We show that cross encoders based on TabSketchFM outperform existing systems on LakeBench related table identification task. We also show that finetuned TabSketchFM can be effectively used to search for tables that are unionable, joinable or are subsets of each other. We highlight that embedding values using sentence encoder models can be a useful addition for some discovery tasks. As far as the search efficiency is concerned, we recommend indexing the datalake offline and at query time only compute embeddings for the query table. This computation is similar to other embedding-based search system.

References

[1] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: Challenges and opportunities," *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 1986–1989, 2019.

[2] A. Khatiwada, R. Shraga, W. Gatterbauer, and R. J. Miller, "Integrating data lake tables," *Proc. VLDB Endow.*, vol. 16, no. 4, pp. 932–945, 2022. [Online]. Available: https://www.vldb.org/pvldb/vol16/p932-khatiwada.pdf

[3] A. Khatiwada, R. Shraga, and R. J. Miller, "DIALITE: discover, align and integrate open data tables," in *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, 2023, pp. 187–190.

[4] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," *Proc. VLDB Endow.*, vol. 11, no. 7, pp. 813–825, 2018.

[5] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller, "JOSIE: overlap set similarity search for finding joinable tables in data lakes," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, 2019, pp. 847–864.

[6] A. Khatiwada, G. Fan, R. Shraga, Z. Chen, W. Gatterbauer, R. J. Miller, and M. Riedewald, "Santos: Relationship-based semantic table union search," *Proc. ACM Manag. Data*, vol. 1, no. 1, p. Article 9, 2023.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[8] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "TURL: table understanding through representation learning," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 307–319, 2020.

[9] H. Iida, D. Thai, V. Manjunatha, and M. Iyyer, "TABBIE: pretrained representations of tabular data," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, 2021, pp. 3446–3456. [Online]. Available: https://doi.org/10.18653/v1/2021.naacl-main.270

[10] 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 '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, 2021, pp. 1780–1790.

[11] P. Yin, G. Neubig, W. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 2020, pp. 8413–8426.

[12] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos, "Tapas: Weakly supervised table parsing via pre-training," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 2020, pp. 4320–4333. [Online]. Available: https://doi.org/10.18653/v1/2020.acl-main.398

[13] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller, "LSH ensemble: Internet-scale domain search," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1185–1196, 2016. [Online]. Available: http://www.vldb.org/pvldb/vol9/p1185-zhu.pdf

[14] A. S. R. Santos, A. Bessa, F. Chirigati, C. Musco, and J. Freire, "Correlation sketches for approximate join-correlation queries," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, 2021, pp. 1531–1544. [Online]. Available: https://doi.org/10.1145/3448016.3458456

[15] G. Fan, J. Wang, Y. Li, D. Zhang, and R. J. Miller, "Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning," *Proc. VLDB Endow.*, vol. 16, no. 7, pp. 1726–1739, 2023. [Online]. Available: https://www.vldb.org/pvldb/vol16/p1726-fan.pdf

[16] K. Srinivas, J. Dolby, I. Abdelaziz, O. Hassanzadeh, H. Kokel, A. Khatiwada, T. Pedapati, S. Chaudhury, and H. Samulowitz, "Lakebench: Benchmarks for data discovery over data lakes," 2023. [Online]. Available: https://arxiv.org/abs/2307.04217

[17] G. Badaro, M. Saeed, and P. Papotti, "Transformers for Tabular Data Representation: A Survey of Models and Applications," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 227–249, 03 2023. [Online]. Available: https://doi.org/10.1162/tacl_a_00544

[18] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1xMH1BtvB

[19] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "Webtables: exploring the power of tables on the web," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 538–549, 2008.

[20] D. Brickley, M. Burgess, and N. Noy, "Google dataset search: Building a search engine for datasets in an open web ecosystem," in *The World Wide Web Conference*, 2019, p. 1365–1375.

[21] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, 2012, pp. 817–828. [Online]. Available: https://doi.org/10.1145/2213836.2213962

[22] O. Lehmberg and C. Bizer, "Stitching web tables for improving matching quality," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1502–1513, 2017. [Online]. Available: http://www.vldb.org/pvldb/vol10/p1502-lehmberg.pdf

[23] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A data discovery system," in *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, Paris, France, 2018, pp. 1001–1012. [Online]. Available: https://doi.org/10.1109/ICDE.2018.00094

[24] A. Bogatu, A. A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, 2020, pp. 709–720.

[25] X. Hu, S. Wang, X. Qin, C. Lei, Z. Shen, C. Faloutsos, A. Katsifodimos, G. Karypis, L. Wen, and P. S. Yu, "Automatic table union search with tabular representation learning," in *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, 2023, pp. 3786–3800. [Online]. Available: https://aclanthology.org/2023.findings-acl.233

[26] Y. Dong, K. Takeoka, C. Xiao, and M. Oyamada, "Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach," in *37th IEEE International Conference on Data Engineering, ICDE 2021*, 2021, pp. 456–467. [Online]. Available: https://doi.org/10.1109/ICDE51399.2021.00046

[27] Y. Dong, C. Xiao, T. Nozawa, M. Enomoto, and M. Oyamada, "Deepjoin: Joinable table discovery with pre-trained language models," *Proc. VLDB Endow.*, vol. 16, no. 10, pp. 2458 – 2470, 2023.

[28] T. Cong, J. Gale, J. Frantz, H. V. Jagadish, and Ç. Demiralp, "Warpgate: A semantic join discovery system for cloud data warehouses," in *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*, 2023. [Online]. Available: https://www.cidrdb.org/cidr2023/papers/p75-cong.pdf

[29] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, 1999, pp. 518–529. [Online]. Available: http://www.vldb.org/conf/1999/P49.pdf

[30] M. Kayali, A. Lykov, I. Fountalis, N. Vasiloglou, D. Olteanu, and D. Suciu, "Chorus: Foundation models for unified data discovery and exploration," *Proc. VLDB Endow.*, vol. 17, no. 8, p. 2104–2114, may 2024. [Online]. Available: https://doi.org/10.14778/3659437.3659461

[31] M. Hulsebos, Ç. Demiralp, and P. Groth, "Gittables: A large-scale corpus of relational tables," *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 30:1–30:17, 2023. [Online]. Available: https://doi.org/10.1145/3588710

[32] 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 *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, 2021, pp. 468–479. [Online]. Available: https://doi.org/10.1109/ICDE51399.2021.00047

[33] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001. [Online]. Available: https://doi.org/10.1007/s007780100057

[34] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, 2018, pp. 19–34. [Online]. Available: https://doi.org/10.1145/3183713.3196926

[35] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra, "Magellan: Toward building entity matching management systems," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1197–1208, 2016. [Online]. Available: http://www.vldb.org/pvldb/vol9/p1197-pkonda.pdf

[36] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer, "A large public corpus of web tables containing time and context metadata," in *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*, 2016, pp. 75–76. [Online]. Available: https://doi.org/10.1145/2872518.2889386

[37] Q. Liu, B. Chen, J. Guo, M. Ziyadi, Z. Lin, W. Chen, and J. Lou, "TAPEX: table pre-training via learning a neural SQL executor," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022. [Online]. Available: https://openreview.net/forum?id=O50443AsCP

[38] T. Yu, C. Wu, X. V. Lin, B. Wang, Y. C. Tan, X. Yang, D. R. Radev, R. Socher, and C. Xiong, "Grappa: Grammar-augmented pre-training for table semantic parsing," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021. [Online]. Available: https://openreview.net/forum?id=kyaIeYj4zZ

[39] F. Wang, K. Sun, M. Chen, J. Pujara, and P. A. Szekely, "Retrieving complex tables with multi-granular graph representation learning," in *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, 2021, pp. 1472–1482. [Online]. Available: https://doi.org/10.1145/3404835.3462909

[40] Y. Suhara, J. Li, Y. Li, D. Zhang, Ç. Demiralp, C. Chen, and W. Tan, "Annotating columns with pre-trained language models," in *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, 2022, pp. 1493–1503.

[41] S. Zhang and K. Balog, "Web table extraction, retrieval, and augmentation: A survey," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 2, pp. 13:1–13:35, 2020. [Online]. Available: https://doi.org/10.1145/3372117

[42] V. Efthymiou, E. Jiménez-Ruiz, J. Chen, V. Cutrona, O. Hassanzadeh, J. Sequeda, K. Srinivas, N. Abdelmageed, and M. Hulsebos, Eds., *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, SemTab 2022, co-located with the 21st International Semantic Web Conference, ISWC 2022, Virtual conference, October 23-27, 2022*, ser. CEUR Workshop Proceedings, vol. 3320. CEUR-WS.org, 2022. [Online]. Available: https://ceur-ws.org/Vol-3320

[43] K. Z. Hu, S. N. S. Gaikwad, M. Hulsebos, M. A. Bakker, E. Zgraggen, C. A. Hidalgo, T. Kraska, G. Li, A. Satyanarayan, and Ç. Demiralp, "Viznet: Towards A large-scale visualization learning and benchmarking repository," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, 2019, p. 662. [Online]. Available: https://doi.org/10.1145/3290605.3300892

[44] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, Ç. Demiralp, and W. Tan, "Sato: Contextual semantic type detection in tables," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 1835–1848, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p1835-zhang.pdf

[45] 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 *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, 2019, pp. 1500–1508.

[46] Y. Deng, C. Chai, L. Cao, Q. Yuan, S. Chen, Y. Yu, Z. Sun, J. Wang, J. Li, Z. Cao, K. Jin, C. Zhang, Y. Jiang, Y. Zhang, Y. Wang, Y. Yuan, G. Wang, and N. Tang, "Lakebench: A benchmark for discovering joinable and unionable tables in data lakes," *Proc. VLDB Endow.*, vol. 17, no. 8, p. 1925–1938, may 2024. [Online]. Available: https://doi.org/10.14778/3659437.3659448

[47] A. Leventidis, M. P. Christensen, M. Lissandrini, L. Di Rocco, K. Hose, and R. J. Miller, "A large scale test corpus for semantic table search," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1142–1151. [Online]. Available: https://doi.org/10.1145/3626772.3657877

[48] E. Zhu and V. Markovtsev, "ekzhu/datasketch: First stable release," Feb. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.290602

[49] T. Acharya, A. K. Ray, and A. C. Gallagher, "*Image Processing: Principles and Applications*," *J. Electronic Imaging*, vol. 15, no. 3, p. 039901, 2006. [Online]. Available: https://doi.org/10.1117/1.2348895

[50] "ECB Statistical Data Warehouse," https://sdw.ecb.europa.eu/, [Online, accessed October 3, 2023].

[51] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, Oct.-Nov. 2018, pp. 3911–3921. [Online]. Available: https://aclanthology.org/D18-1425

[52] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, p. 78–85, sep 2014. [Online]. Available: https://doi.org/10.1145/2629489

[53] Y. Du, Z. Liu, J. Li, and W. X. Zhao, "A survey of vision-language pre-trained models," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, 2022, pp. 5436–5443. [Online]. Available: https://doi.org/10.24963/ijcai.2022/762

[54] W. Zhou, C. Xu, T. Ge, J. J. McAuley, K. Xu, and F. Wei, "BERT loses patience: Fast and robust inference with early exit," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/d4dd111a4fd973394238aca5c05bebe3-Abstract.html

[55] M. Esmailoghli, J. Quiané-Ruiz, and Z. Abedjan, "MATE: multi-attribute table extraction," *Proc. VLDB Endow.*, vol. 15, no. 8, pp. 1684–1696, 2022. [Online]. Available: https://www.vldb.org/pvldb/vol15/p1684-esmailoghli.pdf

[56] 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.*, vol. 42, no. 4, pp. 824–836, 2020. [Online]. Available: https://doi.org/10.1109/TPAMI.2018.2889473

[57] T. Fushiki, "Estimation of prediction error by using *K*-fold cross-validation," *Stat. Comput.*, vol. 21, no. 2, pp. 137–146, 2011. [Online]. Available: https://doi.org/10.1007/s11222-009-9153-8

VIII. Additional Experiments

A. Robustness to hashing implementation

TabSketchFM relies on MinHashes as inputs, but the actual model treats it as a set of input features. To examine how specific the learned weights were to a specific hash implementation, we varied hash algorithms (5 algorithms, including xxhash which we used in our initial learning), with 5 different seeds each, and ran it on two of the benchmarks. Table IX shows the results across 25 different runs. As shown in the table, there is no effect of hashing functions or seeds on the performance on either task, which suggests that the model's weights are not specific to the hash implementation or the seed.

TABLE IX: Effects of hashing implementations and seeds

Dataset	MMH3	FarmHash	SHA	Blake2S	xxhash
ECB Union (R2)	0.91 ±0.02	0.91 ±0.01	0.92 ±0.01	0.89 ±0.02	0.91 ±0.00
ECB Join (F1)	0.88 ±0.01	0.86 ±0.02	0.86 ±0.01	0.85 ±0.03	0.86 ±0.01

B. Need for transformer models

TabSketchFM relies on using a pretrained transformer model that is then finetuned for specific data discovery tasks. An important question is whether we need such a model at all. To address this issue, we performed two separate experiments. In the *first* experiment, we randomly initialized the weights of the pretrained model (Random-PT) and proceeded with 'finetuning' using MinHashes. The results of Random-PT are shown in Table X. For the same two benchmarks, performance with random pretrain weights dropped significantly, with abysmal performance on both tasks indicating that pretraining the model is crucial to learn the data discovery tasks on smaller datasets. In the *second* experiment, we tried to replace the pretrained model with a two layered perceptron (MLP). As

shown in Table X, replacing the transformer with MLP did significantly worse than our method.

C. Model overfitting analysis

Recall that we use early stopping strategy to ensure that our models do not overfit. In addition, k-fold cross-validation analysis can be used to analyze overfitting [57]. We ran k-fold cross-validation analysis on ECB Join and ECB Union, two of the smallest benchmarks (see Table I). In each benchmark, we ran 5-fold analysis with 5 random seeds for each fold i.e. a total of 25 runs. The aggregated F1 Score in ECB Join was 0.85 ± 0.02 and the aggregated R2 Score in ECB Union was 0.90 ± 0.02 which is within the same range as we report in our experiments (Table II). This result supports that our finetuned models are not overfitting.

TABLE X: Effects of pretraining and MinHashes

Dataset	Random-PT	MLP	TabSketchFM
ECB Union (R2)	−263.60	−23.70	0.90
ECB Join (F1)	0.08	0.62	0.86

D. Search

One of the key use cases for the subset search model is to find near duplicates across a data lake. This is useful when customers want to find multiple close copies of the exact same data, to avoid duplicate costs in storage. To examine whether the subset model can be used for this purpose, we created different variants of the 3,072 tables from Eurostat where we either deleted 1, 2, 5, or 10 rows, or we deleted 1, 2, or 3 columns. These tables constituted our corpus. Query tables were the original tables with no deletions.

This task is different from all the tasks used in the paper so far - in the case of finding unionable, joinable or subset tables, we always had a query table as an anchor. In this case, we simply have embeddings for all the tables in the corpus. We formulated detection of near duplicates as finding approximate nearest neighbors (ANNs) over table embeddings problem. ANN computations are extremely fast and efficient. Clustering by embeddings we found was significantly slower, plus clustering requires numerous hyperparameters to control the quality of the clusters. Figure 8 shows the performance of the model as the variants were changed, with perfect precision up to k=5, and a decline to 0.6 at k=10. Table XI shows how many variants were missed in search for the different variants. Not surprisingly, row deletions barely changed performance. Column deletions did influence performance significantly, with a 57% drop once 3 columns were removed. Given that the average number of columns in Eurostat is 10.46, this finding is perhaps not that surprising.

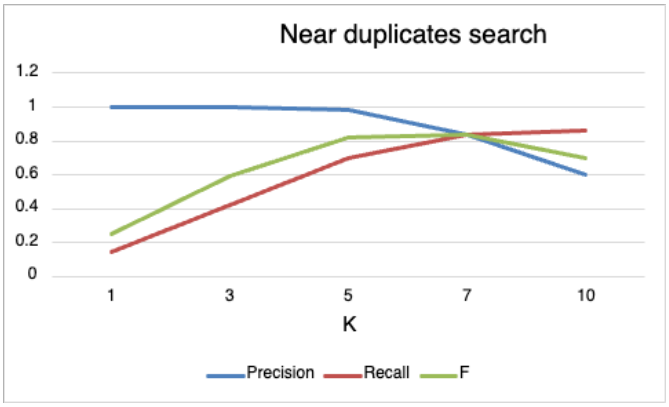


Fig. 8: Performance of TabSketchFM on detecting near duplicates

Variant	Misses%
1 row	0
2 rows	0.0003
5 rows	0
10 rows	0
1 column	0.0846
2 columns	0.3050
3 columns	0.5758

TABLE XI: Miss percentage in search