Pythagoras: Semantic Type Detection of Numerical Data in Enterprise Data Lakes

Sven Langenecker
LÄPPLE AG & DHBW Mosbach
s.langenecker@laepple.de

Christoph Sturm
DHBW Mosbach
christoph.sturm@mosbach.dhbw.de

Christian Schalles
DHBW Mosbach
christian.schalles@mosbach.dhbw.de

Carsten Binnig
Technical University of Darmstadt & DFKI
carsten.binnig@cs.tu-darmstadt.de

ABSTRACT

Detecting semantic types of table columns is a crucial task to enable dataset discovery in data lakes. However, prior semantic type detection approaches have primarily focused on non-numerical data despite the fact that numerical data play an essential role in many real-world enterprise data lakes. Therefore, existing models are typically rather inadequate when applied to data lakes that contain a high proportion of numerical data. In this paper, we introduce *Pythagoras*, our new learned semantic type detection approach specially designed to support numerical along with non-numerical data. *Pythagoras* uses a GNN in combination with a novel graph representation of tables to predict the semantic types for numerical data with high accuracy. In our experiments, we compare *Pythagoras* against five state-of-the-art approaches using two different datasets and show that our model significantly outperforms these baselines on numerical data. In comparison to the best existing approach, we achieve F1-Score increases of around +22%, which sets new benchmarks.

1 INTRODUCTION

Enterprise data lakes contain numerical data. Enterprise data lakes serve as invaluable repositories of diverse data types, enabling organizations to store and manage vast amounts of information [8, 20]. In enterprise data lakes, numerical data plays a dominant role, making up a much larger proportion compared to non-numerical data [17] since they provide *insights into various business domains*, including finance, manufacturing, healthcare, and marketing. Numerical data often contain critical information such as sales figures, production metrics, customer demographics, and financial records. Therefore, it is essential to provide a solution that can automatically detect the correct semantic type of table columns containing numerical values, enabling data analysts and data scientists to find required data for downstream analysis and thus address the dataset discovery problem in data lakes [4, 5, 10, 15, 21].

Semantic typing targets non-numerical data. In order to enable the discovery of data in data lakes and in particular to provide a solution for the associated task of semantic type detection, many solutions using deep learning techniques have been proposed in the past [6, 13, 18, 26, 30]. Unfortunately, all these existing approaches have primarily focused on detecting the semantic type of non-numerical columns, and with that leaving a critical need for innovative approaches that effectively detect

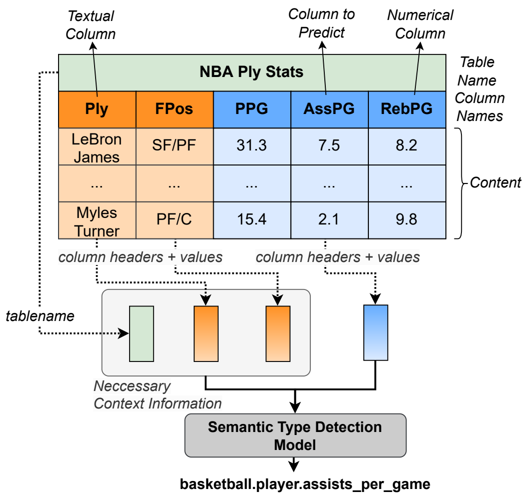


Figure 1: Figure shows an example of predicting the semantic type of the numerical table column 'AssPG'. To predict the correct type, it is crucial for the model to have the possibility to incorporate textual context information such as the table name and neighboring non-numerical columns.

semantic types of numerical columns [17]. The reasons why existing models fall short on numerical data is mainly because of the fact that corpora that were used to train and validate these models primarily contain non-numerical data. Therefore, the models were developed to handle mainly non-numerical data.

Numerical data is much harder. To predict the semantic type of table columns containing numerical values, it is essential to have textual (non-numerical) data of the same table as context information as illustrated in Figure 1. Predicting, for example, the semantic type of the column 'AssPG' by using only the included values {7.5,...,2.1} is almost impossible while values of columns with textual types such as 'Ply' are more indicative for the type. The reason for this is that numerical values have in general a limited information entropy[1] and are often similarly distributed for different semantic content [18].

Context is essential for numerical data. To address this problem, rich non-numerical contextual information, such as the contents of neighboring non-numerical columns, column headers, and table names can be leveraged to increase accuracy in determining the correct semantic type of the numerical column. In the example of Figure 1 the table name 'NBA Ply Stats' and the values of the textual columns ('Ply' and 'FPos') can be leveraged

[1]Generally numerical values can be encoded with much less bits than string values resulting in lower overall entropy values [25]

as context information. This enables the model to recognize that the table belongs to the basketball domain, thereby enhancing its ability to predict the semantic type of the 'AssPG' column as 'basketball.player.assists_per_game'. As such, for a semantic type detection approach designed to handle numerical data, it is crucial to incorporate the capability to leverage all contextual information within the model architecture. Unfortunately, existing model architectures do not have such a predefined technique where non-numeric contextual information can be strategically leveraged for predicting numerical columns.

Semantic type detection for numerical data. In this paper, we thus introduce our approach based on a novel model architecture for semantic typing called *Pythagoras*, which can not only predict the semantic type of non-numerical table columns with high accuracy but also of numerical table columns. To achieve this, the main idea of the new model architecture is to use graph neural networks (GNNs) together with a new graph representation of tables and their columns. This graph representation includes directed edges to provide necessary contextual information (e.g. table name, neighboring non-numerical column values) for predicting the correct semantic type of numerical columns using the GNN message passing mechanism. Thus, the model learns which contextual information is relevant for determining the semantic type. To the best of our knowledge, our semantic type detection model *Pythagoras* is the first approach in this direction.

Contributions of the paper. The main contributions of this paper are: (1) First, we propose a new *graph representation of tables.* In this graph data structure, we use directed edges to model the information flow within tables. Using this graph representation, *Pythagoras* can learn selectively which context information should be taken into account to establish robust predictions on numerical data. (2) As a second contribution, we propose a new *GNN-based neural network architecture* that is able to use our new graph data structure as input and predict the semantic type of table columns. This architecture consists of three main components, (a) a pre-trained language model which encodes call values of tables, (b) a subnetwork to encode an additional specific feature set of the numerical values, (c) and a GNN with a heterogeneous graph convolutional module for aggregating all information and embedding context in the type prediction of columns. (3) Finally, as a third contribution, we show the effectiveness of the graph representation and the model architecture of *Pythagoras* by comparing against five existing state-of-the-art semantic type detection models on two different data lakes that mimic the data distribution of enterprise data lakes and contain tables with non-numerical and numerical columns. The results of this experiment demonstrate that our new model outperforms all baselines on numerical data significantly. To support the integration of our approach into existing applications and to enable further research, we open sourced all our code, data and trained model: https://github.com/DHBWMosbachWI/pythagoras.git.

Outline of the paper. In Section 2 we first introduce *Pythagoras* and our new graph representation of tables. Section 3 details the model architecture. Results and analyses of our experiments are presented in Section 4, followed by a discussion of related work in Section 5. Section 6 concludes the paper.

2 OVERVIEW OF PYTHAGORAS

In the following, we introduce our new semantic type detection approach *Pythagoras*.

2.1 Graph Representation of Tables

Figure 2a demonstrates how we convert a table and its columns into a graph representation using an example table in Figure 2a. The table contains a table name (t_n), two non-numerical columns (c_{nn}), and three numerical columns (c_n), each with column headers (c_h) and column values ($v_1, v_2, ..., v_m$). In the figure, we can see how the table is transformed into a graph $G = \{V, E\}$ composed of a set of nodes V and a set of edges E including four different node types V_{tn}, V_{nn}, V_n, and V_{ncf} for different artifacts.

The first node type V_{tn} (green) represents the tablename. Additionally, the graph contains a node of type V_{nn} (orange) for each non-numerical column. This node type represents the entire column including column values and headers. In the same manner, for each numerical column, we create a node of type V_n (blue) representing numerical columns and their contents. Finally, nodes with a node type V_{ncf} (red) are added for each numerical column to encode specific features of the numerical columns.

We decided to use an additional node type V_{ncf} to encode specific features for numerical columns since this allows us to first use a pre-trained language model (LM) for computing a representation based on the joint features that are shared between both non-numerical and numerical columns such as column headers . In addition, we further add the nodes V_{ncf} for the numerical-only columns, each holding a vector with additional specific features for numerical columns for which we use a separate encoding strategy with a separate simple multilayer perceptron (MLP) network. To be more precise, we additionally encode 192 different statistical features for encoding a numerical column. We publish the full list of features in an extended technical report of this paper[2].

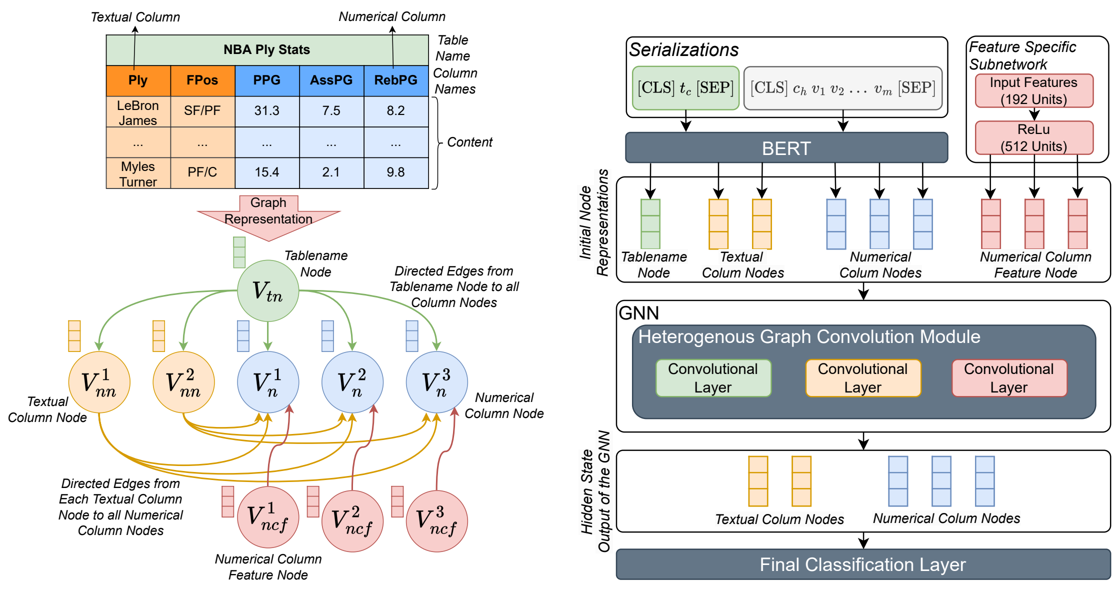
2.2 Leveraging Contextual Information

As described before, only using the numerical values for predicting the semantic type of numerical columns is in general not sufficient, and contextual information is needed. Due to this aspect, we add directed edges to our table graph representation to predefine in which way necessary additional context information should be injected through the message-passing mechanism of GNNs [16] into the numerical column representation (node V_n) and thus enrich it for better predictions.

More precisely, as shown in Figure 2a we construct direct edges from each non-numerical column node V_{nn}^1, V_{nn}^2 to all numerical column nodes V_n^1, V_n^2, V_n^3 (yellow edges) to provide the context information from the non-numerical columns to the numerical columns. Furthermore, we add directed edges in the graph from the table name node V_{tn} to all non-numerical V_{nn} as well as numerical V_n nodes (green edges). This edge handles not only the contextual information for numerical columns but also for non-numerical columns. As we will show in our experiments, using the table name as context information also leads to performance improvements for non-numerical columns. Finally, the graph has directed edges for integrating the additional statistical features into the encoding of numerical columns (red edges from $V_{ncf} \rightarrow V_n$).

As a consequence when using this graph structure as basis for our GNN-based model architecture (cf Section 3), the vector representation computed for the numerical column nodes V_n during training result in an enhanced information content that is more suitable for an accurate prediction of the underline semantic

[2]This technical report can be found at: https://github.com/DHBWMosbachWI/pythagoras.git



(a) Graph Representation **(b) Model Architecture**

Figure 2: (a) Shows the conversion of a table into a heterogeneous graph representation. The key aspect of the graph is that it provides all the necessary contextual information through its structure (nodes and directed edges), resulting in improved predictions of the semantic types of numerical columns. (b) Shows the complete model architecture of the neural network.

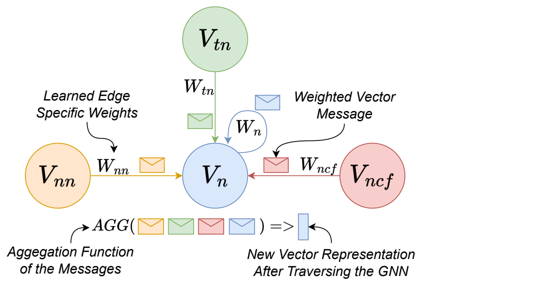

V_{tn}
W_{tn}
Learned Edge Specific Weights
Weighted Vector Message
W_n
V_{nn}
W_{nn}
V_n
W_{ncf}
V_{ncf}
AGG( ) =>
Aggegation Function of the Messages
New Vector Representation After Traversing the GNN

Figure 3: Heterogeneous graph convolutional module of *Pythagoras* for the nodes V_n. Information from the nodes V_{nn}, V_{tn} and V_{ncf} is passed to node V_n. Each edge connection (W_{nn}, W_{tn}, W_{ncf}, W_n) has its own learned weights, which determine how strongly weighted the information is sent over the edges. Finally, all messages (vectors) are combined to form a new representation of the node by an aggregation function.

type. Interestingly, leveraging context by modeling edges in a GNN not only improves the prediction of numerical but also non-numerical types as we show in our evaluation (cf. Section 4).

3 MODEL ARCHITECTURE

In Figure 2b we can see the model architecture of *Pythagoras*. In the following, we explain first the details of the model architecture and then explain how the model can be used to detect numerical semantic types.

3.1 Architecture and Training

The model comprises three essential components. These components include (1) a pre-trained LM to encode all features from non-numerical and numerical columns [3], (2) a specific subnetwork to process the additional features of numerical columns and (3) the GNN to aggregate all information.

The upper part of the architecture in Figure 2b shows how we generate the input of the BERT model to get the initial representations for each column. Additionally, we use BERT to encode table names. To serialize the individual columns, we encode the input sequence for non-numerical as well as for numerical columns, using the column header and the column values as follows: $serialize(c_i)$ ::= [CLS] c_h v_1 v_2 ... v_m [SEP]. Additionally, to generate the initial representation of the node V_{tn} we thus serialize the table name as follows: $serialize(t_c)$ ::= [CLS] t_c [SEP]. For columns and table names, we use the representation computed by BERT for the CLS token as initial node representation for the GNN.

To embed the additional extracted features of the numerical column values, the model contains a feature-specific subnetwork similar to the approach in [13]. As can be seen in the architecture, the subnetwork consists of a linear layer that maps the 192 provided features to a vector that matches the shape of the other initial vector representations (BERT outputs vectors with dimensions of 768). This network is trained end to end with the GNN while the BERT parameters are frozen.

The initial vector representations generated by the BERT model and the subnetwork are used as initial internal representation for all nodes V_{tn}, V_{nn}, V_n, and V_{ncf} in our graph data structure which serves as input for our GNN model. As GNN, we use a heterogeneous graph convolutional module that combines different graph convolutional layers [16] for each occurring edge type. Since we have 3 different edge types in our graph, the heterogeneous convolutional module combines 3 independent

[3]We use BERT but *Pythagoras* is independent of how to generate the initial embeddings, and there may exist alternative language models or embedding methods that could potentially yield even better results in this context.

graph convolutional layers. The heterogeneous convolution module first performs a separate graph convolution on each edge type, then sums the message aggregations on each edge type as the final result for the nodes. Figure 3 shows the behavior of this module for a numerical column node V_n that is connected with other nodes over the different edge types. The module works in a similar way also for non-numerical columns V_{nn} leveraging, however, only information from table name as shown in Figure 2a.

The module allows the model to learn separate weights for the different edge types and thus enables it to embed connected neighboring nodes and their information to different degrees. For example, the model can learn for V_n nodes that the information of the table name (provided by V_{tn}) is less important than the information of adjacent non-numerical columns (provided by V_{nn}). By learning distinct weights for each edge, we can effectively capture the nuances and dependencies in the data, ultimately enhancing the model's ability to make contextually informed predictions that lead to the overall effectiveness of our approach, which we will show more in detail in Section 4.5. After traversing the GNN, we extract the hidden states of the nodes V_{nn} (representing updated non-numerical columns) as well as of V_n (representing updated numerical columns). Subsequently, these hidden states are then fed into a final classification layer to perform the semantic type classification task. In this last classification layer, the output size is determined by the number of distinct semantic types present in the corpus.

3.2 Detecting Numerical Types

To highlight the advantage of using our graph representation of tables together with a GNN for semantic type detection of numerical data types, let us take a look at the following example. Considering the node V_n^1 in Figure 2a which stands for the numerical column 'PPG' (points per game statistic of a basket player) of the table. The column contains values in the range of about 15-32, and its semantic type could be ambiguous. The correct type might be *basketball.player.points_per_game*, *football.player.yards_per_game* or *temperature*).

However, after iterating over a GNN layer, the values of the two non-numerical columns V_{nn}^1, V_{nn}^2 are embedded because of the designed yellow edges. These provide basketball player names (Lebron James, ..., Myles Turner) as well as basketball field positions (SF/PF, ..., PF/C) as context information. According to this additional data, it is clear that the semantic type *temperature* is not very likely for this column. Because of the fact, that tables about player statistics in basketball as well as in football are structured very similarly and contain both columns with player's names and field positions, it is not yet clear whether the semantic type is *basketball.player.points_per_game* or *football.player.yards_per_game* for example.

Besides the previous context data of the non-numerical columns, information about the table name is also injected via the green edges during a GNN layer pass. This information contains the text 'NBA Ply Stats' ('NBA' is the name of the basketball league) and it is now unambiguous determinable that *basketball.player.points_per_game* must be the valid semantic type. The other passed information from the additional statistical feature nodes V_{ncf} also provides an improvement for distinguishing ambiguities, since the value range of numerical columns with different semantic types can be the same but the value distribution can be different. These different characteristics are covered by the extracted statistical features of numerical columns.

Table 1: Characteristics of the datasets in our experiments.

Dataset	#Tables	Non-Num. Cols./Table	Num. Cols./Table	#sem. Types
SportsTables	1,187	2.83	18.1	462
GitTables Numeric	6,577	2.08	8.95	219

4 EXPERIMENTAL EVALUATION

In the following, we first introduce the two datasets SportsTables and GitTables before we describe our experimental setup and evaluation methodology. Afterward, we discuss the main results of our experiments.

4.1 Data Sets and Baselines

Datasets. For evaluating *Pythagoras*, we use two different real-world data lakes with a large number of semantically annotated tables. When selecting the datasets, the goal was to choose a corpora that contain tables with a high proportion of numerical columns. This allows us in particular to explore and compare the existing models with *Pythagoras* on numerical data. As shown in Table 1, we use two corpora SportsTables [17] and GitTables Numeric which is based on [12]. Both corpora contain a high number of numerical columns per table and represents a numerical to non-numeric ratio commonly found in enterprise data lakes [18].

SportsTables [17]. As the first data corpus in our experiments, we use SportsTables. The corpus contains real-world data tables collected from various sports domains such as soccer, basketball, baseball, and football using web scraping techniques. Such data tables are especially rich in numerical columns as many different sport-specific statistical measurements are reported. As can be seen in Table 1, the tables in the corpus contain 2.83 textual and 18.1 numerical columns on average. The corpus includes a very high number of 462 unique semantic types. Thereby semantic types are very fine granular, which is a major challenge for semantic type detection models. For example, there are types such as 'basketball.player.assists_per_game' or 'soccer.player.assists_per_game'.

GitTables Numeric [12]. The original GitTables data set is a corpus of tables created by extracting CSV files from GitHub repositories. Table columns are labeled with semantic types from Schema.org [11] and DBpedia [2] using two different automated annotation methods. In our experiments, we have focused on the annotations origin from DBpedia and the results of the semantic annotation method. For our experiments, we constructed a derived corpus called GitTables Numeric by specifically selecting tables that have a high proportion of numerical columns with the purpose to mimic real-world enterprise data lakes. To achieve this, we only included tables where at least 80% of all table columns are numerical. In order to have enough samples of each semantic type to train, validate, and test the models, we also filtered out columns that have a semantic type occurring less than 10 times in total. Based on these filter criteria, we ended up with a corpus that contains 6,577 tables with 2.08 textual and 8.95 numerical columns per table on average (see Table 1) and a total of 219 semantic types.

Table 2: Experimental results on the SportsTables corpus.

Model	support weighted F1-Score			macro F1-Score		
	numerical	non-numerical	overall	numerical	non-numerical	overall
Sherlock [13]	0.609	0.856	0.641	0.555	0.767	0.57
Sato [30]	0.703	0.961	0.736	0.650	0.903	0.668
Dosolo [26]	0.313	0.822	0.379	0.245	0.782	0.285
Doduo [26]	0.623	0.98	0.67	0.567	0.933	0.594
GPT-3 (fine-tuned)[3]	0.446	0.872	0.501	0.404	0.760	0.423
Pythagoras	**0.829**	**0.996**	**0.851**	**0.790**	**0.97**	**0.803**

Baselines. In our evaluation, we compare our model *Pythagoras* against five state-of-the-art semantic type detection models. As baselines we considered Sherlock [13], Sato [30], Dosolo [26], and Doduo [26]. Despite that Sato and Doduo also incorporate context information to predict the semantic type of a column, they do not specifically address numerical-based columns and do not offer a predefined approach for injecting contextual information into the prediction of numerical columns. All models were trained on the same data as *Pythagoras*.

Given the recent advancements in large language models (LLMs) like GPT-3.5 [3, 22], which have been extensively trained on vast amounts of data, one might wonder if such models cannot predict the semantic type for non-numerical as well as for numerical columns with high accuracy through a straightforward finetuning. Finetuning an LLM to a specific task has already shown success [14, 19, 24, 27]. In light of these considerations, we additionally explore the capabilities of recent LLMs in our study by adding a fine-tuned GPT-3.5 model. We opted for fine-tuning as opposed to prompt designs due to its potential to yield higher performances and to train on a larger number of examples. To build this baseline model we fine-tuned the *gpt-3.5-turbo* model, following the instructions in [1] using the same training data we used for *Pythagoras*.

4.2 Experimental Design

Setup. To run the experiments, we split each dataset into three parts: training, validation, and test set. We divided the datasets into 60% training, 20% validation, and 20% testing splits. Since in both datasets, the gold labels were assigned in an automatic manner by using the individual column headers, we did not include the headers in the serializations of the columns, which is different from what is described in Section 2. When running the experiments, we trained each model using the training split and conducted hyperparameter tuning on the validation set.

In addition, we used the performance results on the validation split during training to apply an early stopping mechanism. To measure the final performance of each model, we loaded the checkpoint of the model with the highest F1-Score on the validation set and then applied it to the test data. We ran each experiment with five different random seeds and reported the mean across multiple runs to obtain statistically reliable results. As evaluation metrics, we used support-weighted F1-Score, weighted by the number of columns per semantic type and the macro average F1-Score as used in previous studies [6, 13, 26, 30].

***Pythagoras* implementation**. We implemented our model *Pythagoras* using Python together with the modules PyTorch [23], DGL [28] and the Transformers library [29]. As described in Section 2, our neural network consists of three main components. A pre-trained LM to generate initial vector representations, a subnetwork for the numerical-based feature set, and a GNN that allows to exchange context information. As pre-trained LM, we used the vanilla BERT [7] (bert-base-uncased) model to be comparable to [26] which comes with 12 encoder layers. We used tokenizer and pre-trained model of the Transformers library from Hugging Face [9]. During the training process, we froze the 12 layers of BERT, preventing their weights from being updated. The graph data structure and the GNN were implemented with the DGL library. To update the weights of the GNN during training, we applied an Adam optimizer with an initial learning rate of 10^{-5} and a linear decay scheduler with no warm-up. Since our purpose is to realize a multi-class prediction task (one semantic type label per column), we used the cross entropy loss as a loss function.

4.3 Exp. 1: Overall Efficiency

Results on SportsTables. Table 2 shows the experimental results on SportsTables. For each model, we list the F1-Scores overall data types to show the total performance, but also the separate average F1-Scores for only numerical and non-numerical data types, respectively. As the first main result, we can see in the table that our model *Pythagoras* outperforms all existing state-of-the-art models in all reported aspects. Looking only at the results on the numerical columns, we can see that our model achieves an improvement of +17.92% support weighted F1-Score and +21.53% macro F1-Score. These results verify that our designed mechanism of providing context information to predict the semantic type of numerical data is more suitable than the methods in the existing models Sato and Doduo.

In Sato, contextual information is provided by a table topic vector, which is formed by a accumulation of all values in the table. Since tables in the SportsTables dataset contains a large proportion of columns with numerical values (on average 18.1 are numerical column and 2.83 are non-numerical, see Table 1), this table topic vector does not have the necessary effect. In addition, Satos linear-chain conditional random field (CRF) also does not lead to significant improvements, since the tables in SportsTables are not always structured in the same way (column orders vary between tables). This aspect can be seen by the comparison of Sato to Sherlock, which is the same model without a table topic vector and a linear-chain CRF module. The improvements from Sherlock to Sato are not significant.

Doduo also achieves only moderate performance values with 0.623/0.564 (support weighted/macro) F1-Score. On one hand, this is due to the fact that only very few individual column values can be included in the token sequence, since the BERT model is limited to 512 elements and the tables have on average 20.93 columns. On the other hand, the BERT model learns the structure

Table 3: Experimental results on the GitTables corpus.

Model	support weighted F1-Score			macro F1-Score		
	numerical	non-numerical	overall	numerical	non-numerical	overall
Sherlock [13]	0.725	0.989	0.775	0.411	0.491	0.707
Sato [30]	0.733	0.991	0.781	0.443	0.707	0.491
Dosolo [26]	0.518	0.986	0.606	0.245	0.694	0.343
Doduo [26]	0.761	0.992	0.804	0.409	0.749	0.489
GPT-3 (fine-tuned)[3]	0.531	0.938	0.610	0.143	0.277	0.211
Pythagoras	**0.813**	**0.990**	**0.846**	**0.476**	**0.893**	**0.544**

of the tables which, as with Sato, has negative effects with non-identical cross-table structures. Furthermore, it is still unclear how deep the understanding of numbers is in LMs like BERT, since they are essentially pre-trained on textual data. Unlike the existing models, our model is independent of the column order of the tables due to the graph structure. If columns are arranged differently between tables, this has no negative effect.

When we examine the results on textual data, we generally observe that all models perform well. In particular, the models Sato, Doduo, as well as our model *Pythagoras* achieve high accuracy. Interestingly, also for non-numerical columns our model is slightly better than existing models with 0.996/0.970 F1-Scores. This improvement is due to the design aspect that our model uses the contextual information of the table name also for the non-numerical column representations ($V_{nn} \rightarrow V_n$ edges). Moreover, our results demonstrate the aspect that on numerical data, the prediction of the semantic type is in general harder than the prediction of non-numerical data.

In summary, the results on the SportsTables dataset demonstrate that our model architecture, in conjunction with the graph representation of tables, leads to significantly improved performance in predicting semantic types for numerical-based columns.

Results on GitTables. Table 3 shows the experimental results on GitTables using the same metrics as before on SportsTables. The results show that *Pythagoras* outperforms all other models in predicting the semantic types. Considering the performance on numerical columns, it becomes evident that our model surpasses the performance of the best existing model, Doduo, by a remarkable improvement of +6.83%/16.38% F1-Score.

This gain in performance highlights the effectiveness of our model in handling numerical data, setting a new benchmark in this domain by outperforming all state-of-the-art approaches. Different from the results on the SportsTables corpus, among the baselines, Doduo and Sato perform nearly equally. This is mainly due to the aspect that the GitTables corpus contains tables with fewer columns on average, and therefore Doduo can use more column values in its token sequence and with that build a better representation using the BERT model.

Looking at the performance on non-numerical data columns, we can see that all models achieve mostly the same support weighted F1-Scores (about 0.990). However, considering the macro F1-Scores our model *Pythagoras* reaches by far the best value with 0.893. This is an improvement to the second-best model Doduo by +19.23%, showing again the benefit of providing the table name as contextual information for predicting the semantic type of non-numerical columns. In summary, the results on GitTables show that our model *Pythagoras* sets new state-of-the-art performances for predicting the semantic type of numerical table columns.

4.4 Exp. 2: Performance for Individual Types

Figure 4 shows a more detailed analysis of the performances between *Pythagoras* and Sato on numerical columns in SportsTables. We chose Sato as comparison model because it was the best baseline model on numerical columns in this dataset. On the left side, the pie chart shows for how many semantic types of numerical columns which model performed better regarding the F1-Score. Out of a total of 384 numerical semantic types, *Pythagoras* was able to achieve substantially better performances than Sato on 202 of them. For 80 types, the two models achieve equal F1-Scores and for 74, Sato is better than *Pythagoras*. This demonstrates that our model is not only more accurate for individual numerical semantic types but also for a very large proportion of them.

To show how large the F1-Score differences between the two models across the numerical types are, boxplots of the differences for the cases *Pythagoras*>Sato and vice versa are shown on the right of Figure 4. In the case where our model achieves higher F1-Scores, we can see that the median value of the distances is 0.2. The 0.75 quantile is 0.4 and there are also a few types where our model is better than Sato by more than 0.9. In addition, the distribution is shifted upwards towards the larger distance values. In the case where Sato is better, the median is about 0.1 and the 0.75 quantil is 0.2. The distribution is also shifted upwards, but not as much as in the other case. In conclusion, these results show that there are many types for which *Pythagoras* performs much better than Sato and Sato can only achieve very low F1-Scores, and the differences to our model are significant. In the other case, for the majority of types in which Sato performs better, our model *Pythagoras* achieves only slightly lower scores.

Overall, this suggests that our model architecture and the method we designed for providing context information are better suited for detecting the semantic type of numerical data.

4.5 Exp. 3: Ablation Study

Different graph variants. To verify the different design aspects of our approach, we tested variants of *Pythagoras*. At first, we tested modifications of our graph representation of tables. In particular, we wanted to investigate which contextual information has which effect on the prediction of the semantic type. Table 4 shows the results of this ablation study by displaying support weighted and macro average F1-Scores on numerical columns. The first row reports the results of using our regular model and graph while the next rows presents the results when various nodes and edges are removed in the graph representation. Here w/o V_{tn} means that in the graph the node representing the table name has been removed and thus also the provision of this context information for the prediction of the semantic type of the columns. Note, that the other nodes V_{nn} and V_{ncf} are still present

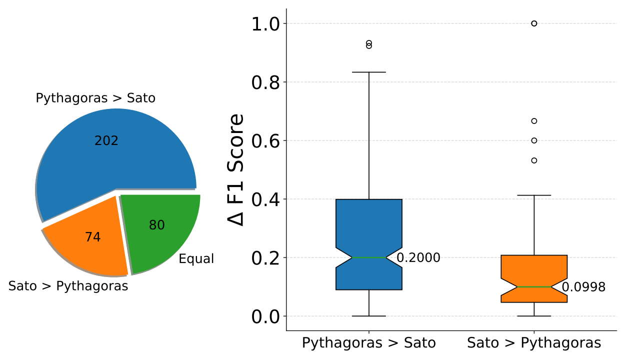


Figure 4: The left chart shows the number of numerical types for which *Pythagoras* performs better than Sato and vice versa. In the right chart we see box plots for the F1-Score differences between the two models on the different numerical semantic types where *Pythagoras* was better than Sato and vice versa.

Table 4: Ablation study results on only numerical columns of the SportsTables dataset. We tested different graph structures that provide different types of contextual information (upper part). The lower part shows results when including the column header c_h as additional information in the serialization of a column.

Variant	support weighted avg F1-Score	macro avg F1-Score
Pythagoras	0.829	0.790
w/o V_{tn}	0.812	0.759
w/o V_{nn}	0.785	0.733
w/o V_{ncf}	0.813	0.765
w/o V_{tn}, V_{nn}	0.724	0.693
w/o V_{tn}, V_{nn}, V_{ncf}	0.324	0.252
w/ original c_h	0.991	0.950
w/ synthesized c_h	0.972	0.926

in the graph and still provide contextual information to the numerical columns representations. Equally, w/o V_{nn} means that the edges of non-numerical to numerical columns have been removed and thus the flow of information from the non-numerical columns no longer occurs during a GNN layer pass. However, in this variant, the other nodes are present.

The first finding that can be seen in the results is that when we remove the nodes V_{nn}, we see the highest performance drop. The F1-Score decreases in this case -0,044/-0,057 in comparison to the regular model. Thus, we can conclude that the most important contextual information for a correct prediction of the semantic type of numerical-based columns are the values of the non-numerical columns from the same table. The second most important context is the table name (V_{tn}) and the least important are the statistical features of the numerical values in the columns (V_{ncf}). Without the table name as context, the model performance decreases a bit more than without the statistical features. To see how good the performance is when making a semantic type prediction only using the numerical values of the columns (V_n and V_{ncf}), we have also considered a variant in which V_{tn} (table name) and V_{nn} (non-numerical columns) nodes are not present. With this variant, the F1-Score drops very sharply and the model only achieves values of 0.724/0.693. This result again shows the immense importance of textual context information in predicting the semantic type of numerical data. In addition, we have tested a variant in which only the V_n nodes are present (w/o V_{tn}, V_{nn}, V_{ncf}). As expected, we just get similar performances to the Dosolo model, since in this constellation both model structures are very similar.

Different column serializations. As mentioned before, in the experiments of Section 4.2, we did not include the original column headers c_h in the serialization of a column because they were previously used to semi automatically assign the true semantic types (gold labels) to the columns. However, to show the impact column headers can have on the performance of numerical column predictions, we created synthetic column headers and used them in an experiment. We created the synthesized column headers using GPT by giving us a list of 10 possible abbreviations for the respective column headers. For example, for the header "Player Age" GPT provided the list ["PA", "PlAge", "PAG", "PLAG", "PlrAge", "PlyAg", "PLA", "PrAge", "PlyrA", "PlayA"]. Afterward, for each column, we randomly selected an abbreviation from the list and used it as the column header. The lower part of Table 4 shows the results of this experiment. We can see that the inclusion of column headers has an additional positive effect on predicting the semantic types of numerical data, achieving F1-Scores of 0.972/0.926 (close to the performance when using the original highly indicative column headers).

5 RELATED WORK

In the following, we will present an overview of existing approaches and discuss the main shortcomings when applied to numerical data.

Columnwise Models. Columnwise models exclusively leverage values from a single column, omitting the inclusion of contextual information from the table. Sherlock [13] is columnwise model which extracts multiple features, such as character distributions, word embeddings, text embeddings, and column statistics from individual columns. These features are then processed through a combination of multi-layer subnetworks and a primary network, which comprises two fully connected layers. Dosolo [26] is a columnwise model that uses the pre-trained BERT model combined with an attached output layer to implement a semantic type detection model. Given that BERT receives token sequences (i.e. text) as input, they convert a column into such a sequence. When serializing the columns, the individual column values are first converted into a string and then concatenated to a sequence.

Tablewise Models. Tablewise models leverage the entire table as input. The advantages of this approach lie in its ability to utilize contextual information from the table, enhancing the precision of semantic type prediction for individual table columns. Building upon Sherlock, Sato is a tablewise model that incorporates Latent Dirichlet Allocation (LDA) features to capture table context and integrates a CRF layer to learn column type dependency. With this, Sato's prediction quality improves over Sherlock. Dosolo & Doduo are both models from [26]. In contrast to Dosolo, Doduo is a tablewise model designed to process an entire table as input to the BERT model. To do this all columns and their values are concatenated one after the other to form an input sequence. The major difference between the two approaches and their serialization techniques is that with Dosolo a column type is predicted independently of other data, whereas Doduo captures the data of neighboring columns to make a prediction of a column semantic type.

Discussion on Existing Approaches. Recent research papers [6, 26, 30] have shown that columnwise models are limited since they can not use context information when predicting the semantic types of columns. As the need for contextual information is even more important for numerical columns, columnwise models are unsuitable for its detection.

As such, Sato [30] and Doduo [26] incorporate also context information like the table-topic or values from neighboring columns as described above. However, these models are mainly designed to handle non-numerical data since used corpora contain almost entirely textual data. If tables contain many numerical and only a few textual columns the context information provided is reduced and the methodologies implemented in Sato and Doduo will not provide the necessary context information to work on numerical data. For example, the table-topic vector of Sato provides no benefit, since numerical values are dominant in the table. With Doduo, the serialization of the table also contains essentially numerical data and this leads to the same effect. However, we show that a controlled and predefined embedding of context information, as we implemented it in *Pythagoras*, leads to a significant improvement on numerical columns. Another shortcoming of Doduo is the limited amount (512 Elements) of column values that can be included when serializing the table. With this, increasing the number of table columns means a decreasing the number of values of each column in the input.

For wide tables, this results in an insufficient column representation and also to only a few values that serve as context information. This is a problem for the prediction on numerical columns, where context information is needed. In our experiments, we demonstrated that our new model *Pythagoras* can handle wide tables containing many numerical columns in a better way. Furthermore, Sato and Doduo have the disadvantage that they essentially learn the order of the columns in the table. This creates a major dependency that tables must have the same column arrangement to ensure the model works adequately. In our opinion, table structures are not always the same, especially in data lakes. For example, Sato's pairwise potentials are learned only for adjacent columns. Whenever there is a different order of the columns, which causes direct neighbors are changed, the learned potentials are no longer useful. Doduo is also sensitive to the column order in a table because the underline BERT model is sensitive to the order in the input sequence. However, our model *Pythagoras* is completely independent of the column order due to the used graph representation of tables together with the GNN architecture.

6 CONCLUSION

The task of semantic type detection of table columns stored in data lakes is crucial to address the dataset discovery problem. Due to the fact that a large proportion of data in enterprise data lakes are numerical [18] and often contains critical information, it is even more important to provide a solution that can detect the underline semantics for these data types robustly. While recent papers propose approaches for extracting semantic types, unfortunately, they have been designed primarily on non-numerical data and therefore do not provide accurate performances when used on numerical data columns. To tackle this problem, we suggested in this paper our new semantic type detection approach *Pythagoras*, specifically designed to robustly handle numerical columns. Our graph representation of tables and GNN architecture establish an intrinsic mechanism that provides all necessary context information to determine the correct semantic type of numerical columns. Experimental results comparing *Pythagoras* against five state-of-the-art models on two different datasets containing mainly numerical table columns show that our approach sets new benchmarks for predicting the semantic type of numerical data.

ACKNOWLEDGMENTS

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) within the "The Future of Value Creation – Research on Production, Services and Work" program (funding number 02L19C150) and by the state of Hesse as part of the NHR program. We also thank hessian.AI, 3AI, DFKI, and DHBW for their support.

REFERENCES

[1] Open AI. 2023. *Fine-tuning*. Open AI. Retrieved October 22, 2023 from https://platform.openai.com/docs/guides/fine-tuning
[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 722–735.
[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.
[4] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. 2019. Dataset Search: A Survey. *The VLDB Journal* 29, 1 (aug 2019), 251–272. https://doi.org/10.1007/s00778-019-00564-x
[5] Christina Christodoulakis, Eric B. Munson, Moshe Gabel, Angela Demke Brown, and Renée J. Miller. 2020. Pytheas: Pattern-Based Table Discovery in CSV Files. In *VLDB*, Vol. 13. VLDB Endowment, 2075–2089. https://doi.org/10.14778/3407790.3407810
[6] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. In *VLDB*, Vol. 14. VLDB Endowment, 307–319. https://doi.org/10.14778/3430915.3430921
[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423
[8] James Dixon. 2014. Data Lakes Revisited. https://jamesdixon.wordpress.com/2014/09/25/data-lakes-revisited/.
[9] Hugging Face. 2023. *bert-base-uncased*. Hugging Face. Retrieved October 22, 2023 from https://huggingface.co/bert-base-uncased
[10] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. 2023. Table Discovery in Data Lakes: State-of-the-Art and Future Directions. In *Companion of the 2023 International Conference on Management of Data* (Seattle, WA, USA) *(SIGMOD '23)*. ACM, New York, NY, USA, 69–75. https://doi.org/10.1145/3555041.3589409
[11] R. V. Guha, Dan Brickley, and Steve Macbeth. 2016. Schema.Org: Evolution of Structured Data on the Web. *Commun. ACM* 59, 2 (jan 2016), 44–51. https://doi.org/10.1145/2844544
[12] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. *Proc. ACM Manag. Data* 1, 1, Article 30 (may 2023), 17 pages. https://doi.org/10.1145/3588710
[13] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *SIGKDD* (Anchorage, AK, USA) *(KDD '19)*. ACM, New York, NY, USA, 1500–1508. https://doi.org/10.1145/3292500.3330993
[14] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How Can We Know What Language Models Know? *Transactions of the Association for Computational Linguistics* 8 (2020), 423–438. https://doi.org/10.1162/tacl_a_00324

[15] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-Based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1, Article 9 (may 2023), 25 pages. https://doi.org/10.1145/3588689

[16] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[17] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. 2023. SportsTables: A new Corpus for Semantic Type Detection. In *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings (LNI, Vol. P-331)*. Gesellschaft für Informatik e.V., 995–1008. https://doi.org/10.18420/BTW2023-68

[18] Sven Langenecker, Christoph Sturm, Christian Schalles Schalles, and Carsten Binnig. 2023. Steered Training Data Generation for Learned Semantic Type Detection. *Proc. ACM Manag. Data* 1, 2, Article 201 (jun 2023), 25 pages. https://doi.org/10.1145/3589786

[19] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. In *VLDB*, Vol. 14. VLDB Endowment, 50–60. https://doi.org/10.14778/3421424.3421431

[20] Christian Mathis. 2017. Data Lakes. *Datenbank-Spektrum* 17, 3 (2017), 289–293.

[21] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. In *VLDB*, Vol. 12. VLDB Endowment, 1986–1989. https://doi.org/10.14778/3352063.3352116

[22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *ArXiv* abs/2203.02155 (2022).

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library.. In *NeurIPS*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett (Eds.). Curran Associates Inc., Red Hook, NY, USA, Article 721, 12 pages.

[24] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. 2019. Language Models as Knowledge Bases?. In *EMNLP-IJCNLP 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 2463–2473. https://doi.org/10.18653/V1/D19-1250

[25] Claude Elwood Shannon. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal* 27 (1948), 379–423. http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

[26] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-Trained Language Models. In *SIGMOD*. ACM, New York, NY, USA, 1493–1503.

[27] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-Trained Transformer is Almost All You Need towards Democratizing Data Preparation. In *VLDB*, Vol. 14. VLDB Endowment, 1254–1261. https://doi.org/10.14778/3457390.3457391

[28] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *ArXiv* abs/1909.01315 (2019).

[29] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. TUTA: Tree-Based Transformers for Generally Structured Table Pre-Training. In *SIGKDD* (Virtual Event, Singapore) *(KDD '21)*. ACM, New York, NY, USA, 1780–1790. https://doi.org/10.1145/3447548.3467434

[30] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. In *VLDB*, Vol. 13. VLDB Endowment, 1835–1848. https://doi.org/10.14778/3407790.3407793