Column Type Detection Based on Pretrained Language Models with Various Column Encodings

Peining LI † and Mizuho IWAIHARA ‡

Graduate School of Information, Production and Systems, Waseda University
2-7 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135 Japan
E-mail: † peining_li@akane.waseda.jp, ‡ iwaihara@waseda.jp

Abstract Real-world tables provide valuable long-tailed facts, and detecting semantic types of table columns is important for table understanding and associated tasks. However, existing methods are often built on heavily-engineered features, such as statistical features and straightforward string matching, which are not robust to dirty data and lack of extensibility. Deep learning models in the field of natural language processing have been successfully adopted to various sequence prediction tasks. In this paper, we discuss utilizing pretrained language model BERT and its variants for table column typing. We present a number of table encoding methods that serialize the target table into a token sequence, which is used for finetuning BERT toward the column typing task. Various orderings of table cells are discussed, where Column Encodings are utilized to indicate boundaries of columns and rows. Experimental results show that our model greatly outperforms the state-of-the-art method on both weighted- and macro-averaged F1 scores.

Keyword Column Type Detection, Deep learning, Natural Language Processing, BERT, Pretrained Language Models

1. INTRODUCTION

A variety of data management operations (e.g., data cleaning, data integration and information retrieval tasks) require meta information about tables, and detecting data column types is one of the most basic but important tasks. In many business systems, the column types of a table are necessary for having a good understanding about the table for supporting user's operations. For example, through detecting the column types, Google can extend its support for tables [11] and Tableau [15] can provide a more appropriate way of visually presenting tabular data.

For a given table, we first assume a vertical table has horizontal headings, which means the column name cells are on the top of the table and the corresponding data are displayed in columns, so that the data in different cells in one column are belonging to the same filed. On the other hand, a horizontal table has column name cells on one side of the table and rows of corresponding data on the other, we can transpose it so we can also get a vertical one.

For one column in a vertical table, each non-empty cell is assumed to belong to common types. Column types are generally classified into atomic types and semantic types. Atomic types represent value domains of the cells, such as string, boolean and integer [13]. Semantic types, on the other hand, represent semantic information indicating what area of information the cells are representing, such as person name, location, and description.

Detecting the semantic types of target table columns is to match the common types of cells with semantic classes [1] defined in knowledge graphs. Figure 1 shows an example of a target table, where missing column types need to be detected. The table in Figure 1(a) is a list of person names and is actually a part of Figure 1(b). Figure 1(b) displays a table for data of best-selling soundtrack albums, with properties "*rank*", "*year*", "*album*", "*artist*" and "*sales.*" In Figure 1(a), it may be difficult to predict the semantic types because of the lack of context information, and each of "*person,*" "*writer*" and "*artist*" could be right answers. However, in Figure 1(b), we can obtain richer information from the other columns in the same table to help determine the correct column types.

To leverage the context information of tables, in this paper we propose a column type prediction model which is trained on token sequences generated by serializing tables into token sequences. Our proposed model is based on BERT [3], one of the pre-trained language models (LMs) and have been adopted into many tasks of NLP field. We here formulate the task of column type detecting as a multi-class classification problem.

For a given table, we serialize the table through our novel table encoding methods, where Column Encodings are introduced for distinguishing the boundary of rows and columns. Our table encoding methods utilize information of the cells around the target column for predicting its semantic type. The details of model architecture will be explained in Section 4.

???
Whitney Houston & Various
Bee Gees & Various
Various
James Horner & Various

(a)

???	???	???	???	???
1	1992	The Bodyguard	Whitney Houston & Various	45,000,000
2	1977	Saturday Night Fever	Bee Gees & Various	40,000,000
3	1987	Dirty Dancing	Various	32,000,000
4	1997	Titanic: Music from the Motion Picture	James Horner & Various	30,000,000

(b)

Figure 1: Two examples of vertical tables with missing semantic types. (a) is a table with only one single column, and (b) is a vertical table with multiple columns. There is one column in table (b) that has identical values with the table (a), and we consider the task to detect semantic types of these two columns separately. It is difficult to resolve the ambiguity of the column and get fine-grained correct semantic types in (a), but we can get more information in (b) and improve the accuracy of column type detection.

Our main contributions can be summarized as below:

1. Our proposed methods outperform the state-of-the-art in the column type detection task. Our best model has raised the macro and weighted average F1 scores by 0.048 and 0.016, respectively.

2. Our proposed method utilizes the pretrained language model BERT for generating column representations from serialized cell sequences. This leads to better extensibility than previous studies, which generally rely on heavily-engineered task-specific features [2].

3. We introduce a number of novel table encoding methods to incorporate context rows and columns through various input and Column Encoding methods. This paper is the first that the notion of cross-column input encodings is applied to the column type detection task.

2. RELATED WROK

In the field of table understanding, most of existing works rely on rule-based or probabilistic methods, having difficulties in extensibility [2]. Regarding the rule-based approach, Heist et al. (2021) [5] asserted types of listings and exploited contextual relations between the subject entities of a listing and the listing context. Goel et al. (2012) [4] used conditional random fields (CRFs), one of the statistical modeling methods that is often used in prediction tasks and pattern recognition, and predicted the column types of tables. Limaye et al. (2010) [9] applied a graphical model for predicting entities on values, detecting column types, and annotating relationships between columns simultaneously.

Recently, researchers also incorporate deep learning models such as convolutional neural network (CNN) into column typing tasks [12].

Wang et al. (2021) [14] propose a novel relational table representation learning approach, using both intra- and inter-table contextual information, for determining the type of each column and the relation between columns. Hulsebos et al. (2019) [7] proposed SHERLOCK, a deep learning model which extracts features from column values, such as statistical properties, character distributions, word embeddings, and paragraph vectors, and use these features to predict column semantic types. Based on Sherlock, Dan Zhang et al. (2019) [15] developed a hybrid model to predict the semantic types of columns in tables, called SATO, which incorporates the global context and local context in a table for column type prediction. Deng et al. (2020) [2] proposed a structure-aware Transformer-based model called TURL, pretrained on relational Wikipedia tables. Finetuned on 6 downstream tasks, TURL shows better results than earlier task-specific methods, but it requires a number of external information about tables.

In the field of natural language processing, Bidirectional Encoder Representations from Transformers (BERT) developed by Devlin et al. (2018) [3] is one of the most widely used pretrained language models and has achieved remarkable results in various downstream tasks. Based on BERT, Liu et al. (2019) [11] presented a set of BERT design and training strategies and developed RoBERTa. The modification is simple, but it achieves good results in a variety of downstream tasks. Also based on BERT, Lan et al. (2019) [8] developed ALBERT, which is a lite BERT for self-supervised learning of language representations [8] and introduced factorized embedding parameterization. Liu (2019) [10] proposed BERTSUM, utilizing interval Segment Embeddings for extractive summarization.

3. PROBLEM FORMULATION

We assume that a target table is given only with cell

values and column names, other information such as the table title, chapter title, or other supplemental information is not available. From this situation of having minimum information of the target table, our goal is to predict semantic types of the table columns. We formulate it as a multi-class classification task, predicting one of the predefined semantic types as the type of the target column.

We consider a set of tables as dataset T, which consists of n tables $T^1, T^2, \ldots, T^n$, and a set of labels for possible semantic types denoted as S.

For each table t in dataset T, the columns of table T^t are denoted as $c_1^t, c_2^t, \ldots, c_m^t$, and the true semantic type of these columns are denoted as $s_1^t, s_2^t, \ldots, s_m^t$, where each type belongs to the predefined label set S, i.e. $s_{t,i} \in S$.

Given a table T^t with columns $c_1^t, c_2^t, \ldots, c_m^t$, and the vocabulary of column types S, the column type prediction by model M is to determine a column type $M(T^t, c_i^t) \in S$ that best describes the semantics of each c_i^t.

4. PROPOSED ENCODING METHODS

Our proposed model architecture is shown in Figure 2. We conduct cross-column encoding to serialize the input table, supply it into the pretrained language model layers, and obtain the precited column semantic type as the result. The novelty is that we propose a number of table encoding methods to serialize the target table into a token sequence, which is used for finetuning BERT toward the column typing task.

The core of our column encodings is segment embeddings introduced to distinguish the boundary of columns and rows, so different values are assigned. In the following, the target column is denoted as column A. Another column, called associated column, denoted as B, is arbitrary chosen from the same table for providing inter-column contexts. The cells of columns A and B are transformed into token embeddings by the tokenizer of BERT. The tokens from columns A and B are distinguished by assigning column embeddings E_A and E_B, respectively, to these tokens in segment embeddings.

Special Tokens. We utilize three types of special tokens of BERT, namely [CLS], [SEP], [PAD], for input token sequences.

[CLS]: The [CLS] token always appears as the first token in the input sequence. In our case, [CLS] is at the beginning and just before the first value token of the target column A. Segment embedding E_A is assigned to the [CLS] token. When [CLS] appears, BERT will watch the whole

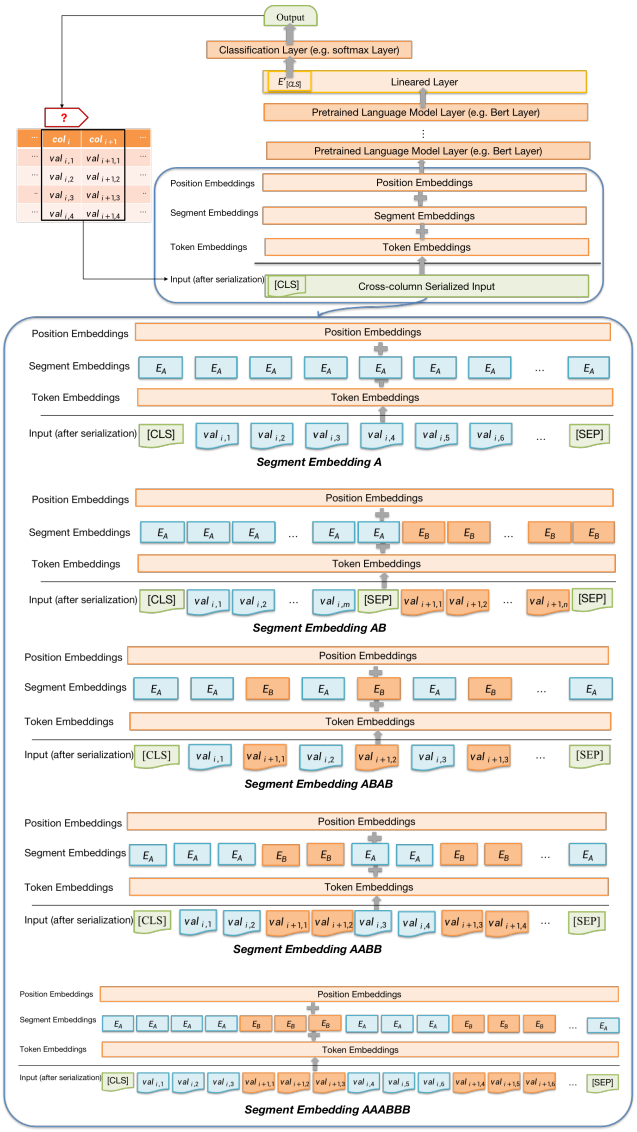


Figure 2: Model structure. There are five different input formats, and we assign different column encoding methods to them.

information of the input and the output of this token will be used in classification tasks.

[SEP]: The [SEP] token is inserted at the end of the target column. For the encodings that alternate between values of target column A and its associated column B, [SEP] is inserted as the last token. The segment embedding for a [SEP] token is corresponding with the column this [SEP] belongs to. The [SEP] token is to let BERT distinguish the finish of multiple column inputs. Therefore, in the segment embeddings, the segment embedding tokens will change when a [SEP] token is encountered.

[PAD]: The [PAD] token appears when padding or there is no following associated column, and segment embedding value E_A is assigned to this token. This is to make the

inputs of different length of columns have the same length of embedding tokens.

Here, we discuss five methods for column encoding, which are illustrated in Figure 2. In the following, we denote the cell value of row j in column c_i as $val_{i,j}$.

Column Encoding A: Normally, the BERT input should be like a sentence, so for a column, we can simply align all cells in target column as input.

For target column A in a given table, we only consider the column c_i as input. So we serialize c_i as

$$\text{serialize}_{\text{A}}(c_i) ::= [\text{CLS}], val_{i,1}, \dots, val_{i,m}, [\text{SEP}]$$

Here, we assign the same segment embedding E_A on all input tokens, meaning that all the input tokens are in the same column.

$$\text{segment}_{\text{A}}(c_i) ::= E_A, E_A, \dots, E_A, E_A$$

Column Encoding AB: Column Encoding A is fine for single target column (A) input, but cannot use the information of context column (B), so we propose Column Encoding AB to deal with this problem.

Column c_i of target column A, and column of its associated column B (here we assume it as c_{i+1}) are serialized as input. We concatenate these two columns simply by one column after the other as follows:

$$\text{serialize}_{\text{AB}}(c_i, c_{i+1}) ::$$
$$= [\text{CLS}], val_{i,1}, \dots, val_{i,m}, [\text{SEP}], val_{i+1,1}, \dots, val_{i+1,n}, [\text{SEP}]$$

We assign segment embedding E_A to tokens in column c_i, which is the target column, and assign E_B to column $\text{c}_{\text{i+1}}$. So we define

$$\text{segment}_{\text{AB}}(c_i, c_{i+1}) ::= E_A, E_A, \dots, E_A, E_A, E_B, \dots, E_B, E_B$$

Column Encoding ABAB: To further use the relation of cells in the same row, rather than simply concatenating A and B, we propose Column Encoding ABAB.

Column Encoding AB can capture row-wise contexts of the target column and associated column, but the column-wise contexts between A and B are captured only through separated two groups of tokens. Column Encoding ABAB in the following aligns cells of columns A and B in the same row side-by-side, so that the model can be aware of tuple-wise value associations between A and B. In the following, we order cells of the target column A and the associated column B in the same row alternately:

$$\text{serialize}_{\text{ABAB}}(c_i, c_{i+1}) ::= [\text{CLS}], val_{i,1}, val_{i+1,1}, val_{i,2}, val_{i+1,2}, \dots, [\text{SEP}]$$
$$\text{segment}_{\text{ABAB}}(c_i, c_{i+1}) ::= E_A, E_A, E_B, E_A, E_B, \dots, E_A$$

Column Encoding AABB: To further change the proximity weights on the target cell and its neighboring row and column, we propose Column Encoding AABB.

We group two neighboring cells of the target column c_i and its associated column $\text{c}_{\text{i+1}}$, and alternately concatenate the two-cell groups of c_i and $\text{c}_{\text{i+1}}$ as the input sequence. This encoding is intended to capture relatedness between the target cell and its neighboring row and column, with equal proximity in the input sequence. We define

$$\text{serialize}_{\text{AABB}}(c_i, c_{i+1}) ::= [\text{CLS}], val_{i,1}, val_{i,2}, val_{i+1,1}, val_{i+1,2} \dots, [\text{SEP}]$$
$$\text{segment}_{\text{AABB}}(c_i, c_{i+1}) ::= E_A, E_A, E_A, E_B, E_B, \dots, E_A$$

Column Encoding AAABBB: To give even more different proximity weights on A and B, here we propose Column Encoding AAABBB.

Similar to Column Encoding AABB, we group three consecutive cells in target column c_i and its associated column $\text{c}_{\text{i+1}}$, and append the three-cell groups alternatively. This encoding is intended to allocate more cells from the same column than Column Encoding AABB, while retaining proximity to the associated column. We define:

$$\text{serialize}_{\text{AAABBB}}(c_i, c_{i+1}) ::$$
$$= [\text{CLS}], val_{i,1}, val_{i,2}, val_{i,3}, val_{i+1,1}, val_{i+1,2}, val_{i+1,3}, \dots, [\text{SEP}]$$
$$\text{segment}_{\text{AAABBB}}(c_i, c_{i+1}) ::= E_A, E_A, E_A, E_A, E_B, E_B, E_B, \dots, E_A$$

For each column encoding, if target column c_i is the last column in the table, or there is only one column, then we concatenate c_i with a padding column:

$$\text{serialize}(c_i)::$$
$$= [\text{CLS}], val_{i,1}, val_{i,2}, \dots val_{i,m}, [\text{SEP}], [\text{PAD}], \dots [\text{PAD}], [\text{SEP}]$$
$$\text{segment}(c_i) ::= E_A, E_A, E_A, \dots, E_A, E_A, E_A, \dots, E_A, E_A$$

With above five column encodings, we can obtain corresponding position, segment, token embeddings, then sum them up and enter into a pretrained language model layers, like BERT.

We formulate the task of assigning column types as a multi-class classification, such that one of the column types in possible semantic types S is predicted as the semantic type of the target column. On the last layer of the pretrained language model, a multi-class classifier is added as task specific layers. Here we adopt the SoftMax function, for producing the final predicted type as result. In the training, the cross-entropy loss function is applied.

5. EVALUATION

We build our models and compare with the state-of-the-art model SATO as well as other baseline pretrained models.

Dataset. Similar to earlier work SATO [15], we use the table datasets derived from VizNet corpus [6], canonicalize the column names of tables, select columns that have valid column names appeared in predefined 78 common semantic types [15], remove non number or non-English language strings, and the remaining valid tables are used as our benchmark dataset. We obtain 78693 tables in total, with total number of 120472 columns. The number of columns

	Model	macro avg F1 score	weighted avg F1 score
state-of-the-art	SATO	0.756	0.902
Pretrained Model + Different Column Encodings + SoftMax as Classification Layer	**BERT+ABAB**	**0.804 (+0.048)**	**0.918 (+0.016)**
	BERT+AABB	0.788 (+0.032)	0.917 (+0.015)
	BERT+AAABBB	0.792 (+0.036)	0.914 (+0.012)
	BERT+AB	0.779 (+0.023)	0.917 (+0.015)
Pretrained Model + Column Encoding A + SoftMax as Classification Layer	BERT+A	0.763 (+0.007)	0.899 (-0.003)
	RoBERTa+A	0.728 (-0.028)	0.884 (-0.018)
	DistilBERT+A	0.684 (-0.072)	0.879 (-0.023)
	XLNet+A	0.705 (-0.051)	0.877 (-0.025)
	ALBERT+A	0.502 (-0.254)	0.736 (-0.166)
Pretrained Model + Column Encoding A + Machine Learning as Classification Layer	Bert + LogisticRegression	0.559 (-0.197)	0.774 (-0.128)
	Bert + DecisionTreeClassifier	0.412 (-0.344)	0.647 (-0.255)

Table 1: Performance of baselines and our methods. In our experiment, when only input target column and do not consider context information, or use machine learning methods as classification layer, the results are not as good as the state-of-the-art SATO model. However, after introducing different Column Encodings, we can outperform the SATO model.

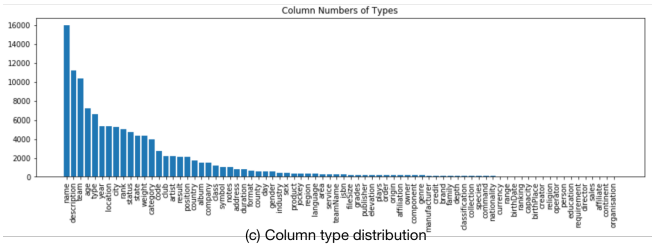
of tables in the dataset is ranging from 1 to 6, and among these column numbers, 1-column tables have the most counts. Regarding rows, the tables have min row number of 1, max row number of 6904, and tables with 2 rows have the most counts. The detailed statistics on the dataset is shown in Figure 3.

Table with col num	1	2	3	4	5	6
Table num	47600	23793	4629	1959	709	3

(a) Table column number distribution

Table with row num	2	3	5	4	10	25	6	7	8	9
Table num	11490	8717	7273	6112	6063	3418	3305	2948	2203	1959

(b) Top10 tables with row number distribution



(c) Column type distribution

Figure 3: Dataset statistics. (a) is the total distribution of table column number. (b) shows the table row numbers that are the top-10 ranked by the table counts. (c) is the distribution of column semantic types, which is long-tailed.

Baselines. SATO is a hybrid machine learning model for annotating table column semantic types, and is the state-of-the-art method. SATO integrates two models on topic modelling and structured prediction modelling [15], to perform column type prediction thorough semantic and structural embeddings of tables. Besides SATO, we also compare with models on variants of BERT. These pretrained language models are already been pretrained on large corpus and can be applied in various downstream tasks through finetuning. These models are widely adopted into NLP field, but rarely used in table inference tasks. In our experiments, BERT is the basic pretrained model, but for comparison we also compare with RoBERTa, DistilBERT, XLNet and ALBERT, where the target column is encoded by Column Encoding A. Also, we compare different machine learning classification layers, logistic regression and decision tree classifier, conducted over these pretrained models as baselines.

Experimental Settings. We use the default BERT 12 layers, cross-entropy loss function, and Adam optimizer with learning rate as 1e-4, and train epochs set to be 20, input max length as 64, and batch size as 300. For input, since certain cells have too long values, it is impractical and unreasonable to input all the values, so for each cell we only use the first 3 words to represent the whole cell, and extend to 5 words when no following context column.

Evaluation Metrics. We use macro-averaged and weighted-average F1 scores for evaluating the results of column type prediction.

Macro average F1 score: Macro-averaged F1 score is calculated as arithmetic mean of F1-score for each type, so it is an unweighted average F1 score. This score treats all types as equal, without taking imbalance into account, so this index is sensitive to long-tailed types.

Weighted average F1 score: Weighted average F1 score is the average of the F1 scores of each semantic type weighted by the number of columns for each type. Compared to macro-averaged F1 score, weighted average F1 score is based on weighing proportional to the column populations in the semantic types.

6. RESULTS

Table 1 shows the performance of our methods and baselines over the dataset. We implement the proposed column encoding methods, on the five pretrained models, and the two classification layers separately. The results show with default Column Encoding A, BERT model gets slightly better macro avg F1 score than the state-of-the-art model SATO, but worse in weighted avg F1 score. This shows the advantages in introducing pretrained language BERT model. And with our proposed croos-column encoding methods applied on BERT, all of models achieve better performance than the state-of-the-art model SATO. Among all the results, the BERT with ABAB Column Encoding performs best both in macro-average F1 score and weighted average F1 score, which are marked in Table 1. When using BERT as the base pretrained model, SoftMax as classification layer, and Column Encoding ABAB, the macro- and weighted-average F1 scores of 0.804 and 0.918, respectively, are obtained.

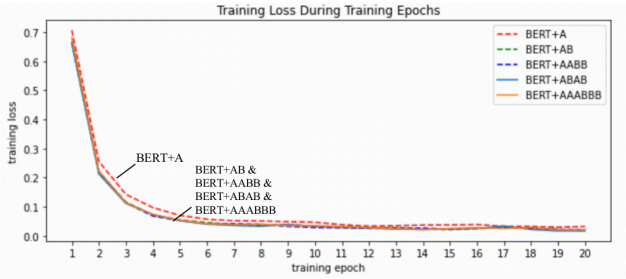


Figure 4: Training loss during training epochs.

To compare the differences of the Column Encoding methods, Figure 4 shows the graph of training loss. There are roughly two trends of the five column encodings. Column Encoding A is getting higher train loss in each epoch, and the other four cross-column methods (AB, ABAB, AABB, AAABBB) are lowering train loss faster. Consequently, when introducing contexts of one associated column, the model can achieve lower loss. Combining with the results in Table 1, we can see our proposed cross-column encoding methods (AB, ABAB, AABB, AAABBB) are achieving lower training loss and higher F1 score than no cross-column methods (A). This shows the rationality and potential for further improving cross-column methods in order to incorporate the context information of the target column for improving accuracy of semantic type prediction.

7. CONCLUSION

In this paper, we proposed five column encoding methods to serialize table cells into a token sequence, to be used for finetuning pretrained language models for semantic table column typing. Column encodings can integrate adjacent column cells to capture their relatedness. Special tokens are utilized to distinguish the boundary of columns and rows. Our results on the column typing task are superior on weighted- and macro-averaged F1 scores than the state-of-the-art model SATO. Also, the training loss graph shows that our cross-column encoding methods can generate lower loss than traditional methods without cross encoding, which shows the rationality of our methods and is promising for further research.

Here is the agenda for our further improvement.

Further analyze the performance on per-type columns and the mechanism behind it. By now, we have known that our model outperforms SOTA model, but we need to further analyze the results, like in what kinds of data our model can perform better or worse than SOTA, and find the mechanism behind them.

Extending the associated columns. Here we only consider one associated column as table context, so in the future we can extend it to a column window with the target column as core. For example, we set the column window size to 3, and use two context columns for better predicting the semantic types of target column. Also, we can set the column window to the max column length of the table, so we can consider all the other columns, but this approach is restricted by the input length limitation of BERT.

Dealing with table columns with too many rows. For the tables having a large number of rows, currently we only use rows that fit into one sequence. In the future, it is possible to use sampling or partitioning one table into subtables, and column-type prediction is performed each subtable separately, and then combining the results of all predictions into the final prediction for the whole table. Besides, a more sophisticated way to achieve this goal would be hierarchically producing embeddings from subtables, and a Transformer in the higher layer which attends over the whole table.

References

[1] J. Chen, E. Jiménez-Ruiz, I. Horrocks, and C. Sutton, "Colnet: Embedding the semantics of web tables for

column type prediction", Proc. of AAAI, pp. 29-36. 2019.

[2] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "Turl: Table understanding through representation learning", arXiv preprint, arXiv:2006.14806, 2020.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding", Proc. of NAACL-HLT, pp. 4171-4186, 2019.

[4] A. Goel, C. A. Knoblock, and K. Lerman, "Exploiting structure within data for accurate labeling using conditional random fields", Proc. of ICAI, pp. 1, 2012.

[5] N. Heist, and H. Paulheim, "Information extraction from co-occurring similar entities", Proc. of the Web Conference, pp. 3999-4009, 2021.

[6] K. Hu, N. Gaikwad, M. Bakker, M. Hulsebos, E. Zgraggen, C. Hidalgo, T. Kraska, and G. Li, "Viznet: Towards a large-scale visualization learning and benchmarking repository", Proc. of ACM CHI, pp. 1-12, 2019.

[7] M. Hulsebos, K. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, C. Demiralp, T. Kraska, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection", Proc. of ACM SIGKDD, pp. 1500-1508, 2019.

[8] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations", Proc. of ICLR, 2020.

[9] G. Limaye, S. Sarawagi, and S. Chakrabarti. "Annotating and searching web tables using entities, types and relationships", Proc. of VLDB Endow 3 (1), pp. 1338-1347, 2010.

[10] Y. Liu, "Fine-tune BERT for extractive summarization", CoRR, abs/1903.10318, 2019.

[11] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach", arXiv preprint, arXiv:1907.11692, 2019.

[12] Y. Suhara, J. Li, Y. Li, D. Zhang, C. Demiralp, C. Chen, and W. Tan, "Annotating Columns with Pre-trained Language Models", arXiv preprint, arXiv:2104.01785, 2021.

[13] P. Venetis, A. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, and C. Wu, "Recovering semantics of tables on the web", Proc. of VLDB Endow. 4 (9), pp. 528-538, 2011.

[14] D. Wang, P. Shiralkar, C. Lockard, B. Huang, X. Luna Dong, and M. Jiang, "TCN: Table Convolutional Network for Web Table Interpretation", Proc. of the Web Conference, 2021.

[15] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, C. Demiralp, and W. Tan, "Sato: Contextual semantic type detection in tables", Proc. VLDB Endow. 13 (12), pp. 1835–1848, 2020.