Numerical Tuple Extraction from Tables with Pre-training

Qingping Yang[†]
Yixuan Cao[†]
yangqingping17s@ict.ac.cn
caoyixuan@ict.ac.cn
Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
Beijing, China

Ping Luo[*†]
Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
Beijing, China
Peng Cheng Laboratory
Shenzhen, China
luop@ict.ac.cn

ABSTRACT

Tables are omnipresent on the web and in various vertical domains, storing massive amounts of valuable data. However, the great flexibility in the table layout hinders the machine from understanding this valuable data. In order to unlock and utilize knowledge from tables, extracting data as numerical tuples is the first and critical step. As a form of relational data, numerical tuples have direct and transparent relationships between their elements and are therefore easy for machines to use. Extracting numerical tuples requires a deep understanding of intricate correlations between cells. The correlations are presented implicitly in texts and visual appearances of tables, which can be roughly classified into *Hierarchy* and *Juxtaposition*. Although many studies have made considerable progress in data extraction from tables, most of them only consider hierarchical relationships but neglect the juxtapositions. Meanwhile, they only evaluate their methods on relatively small corpora. This paper proposes a new framework to extract numerical tuples from tables and evaluate it on a large test set. Specifically, we convert this task into a relation extraction problem between cells. To represent cells with their intricate correlations in tables, we propose a BERT-based pre-trained language model, TableLM, to encode tables with diverse layouts. To evaluate the framework, we collect a large finance dataset that includes 19,264 tables and 604K tuples. Extensive experiments on the dataset are conducted to demonstrate the superiority of our framework compared to a well-designed baseline.

CCS CONCEPTS

• **Information systems** → **Data extraction and integration**; **Information extraction**.

KEYWORDS

tuple extraction, tabular representation, pre-training

ACM Reference Format:
Qingping Yang, Yixuan Cao, and Ping Luo. 2022. Numerical Tuple Extraction from Tables with Pre-training. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539460

[*]Corresponding author.
[†]Also with Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS).

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539460

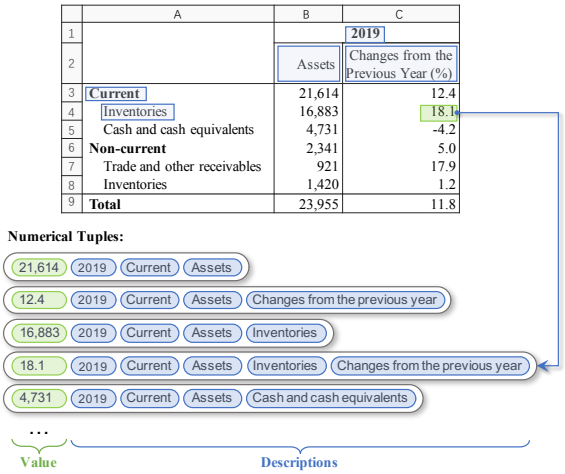
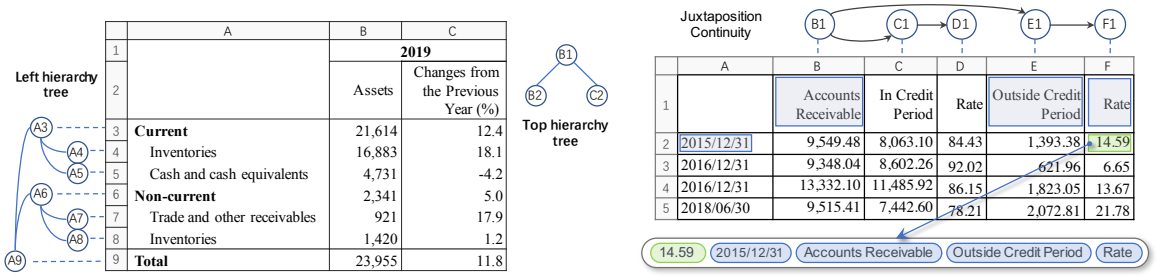


Figure 1: An example illustrates the numerical tuples in a table. The green boxes indicate the values; the blue boxes indicate the descriptions. Best viewed in color.

1 INTRODUCTION

As a semi-structured data structure for efficiently organizing, presenting, and analyzing a range of data, tables are omnipresent on the Web and vertical domains. There are more than tens of billion tables on the Web [2]. A study shows over 750 million users of Microsoft Excel, who produce massive amounts of tables in enterprise environment [11]. Tables are typically treated as a flexible database to store high-dimensional data or present data in published documents in vertical domains, such as finance and academia. According to the statistics from Li et al. [17], of the 270 IPO prospectuses they collected, there were 227 tables per prospectus and 0.5 tables per prospectus page on average. Of the 762 auditor reports they collected, there were 65 tables per report and 1.38 tables per report page on average. Consequently, a tremendous amount of important data are stored in tables.

Unlike the data stored in machine-friendly formats such as relational databases, the data stored in general tables are difficult to understand and apply to downstream tasks. To make tables compact and intuitive for humans, table creators typically project high-dimensional data to two-dimensional layouts by leveraging visual grammar (e.g., indentation, font style, merged cell) [24]. It



(a) Top and Left Hierarchy Trees

(b) Juxtaposition of Cells

Figure 2: An example illustrates the relationships between cells. Best viewed in color.

brings substantial flexibility to the table layout. However, such flexibility poses a considerable challenge in automatically parsing tables. Most modern powerful data manipulation tools and table understanding models make a strong assumption that the table is relational [1, 7, 14, 25]. In addition, converting arbitrary tables into relational data requires a massive investment in table layouts and specific scripts. As a result, these valuable data are still locked in the flexible tables.

One critical step to understanding data in tables is extracting numerical data from tables. The semantics of a numerical data point can be expressed as a numerical tuple consisting of a value and several descriptions. Furthermore, a table can be parsed into a set of numerical tuples, as shown in Figure 1. Numerical tuples contain the whole meaning of data. Moreover, they are organized in a relational format with direct and transparent relationships between elements, making them easy to use for machines. However, the elements of a numerical tuple are usually scattered among several cells. For example, to understand the cell C4 in Figure 1, we need to consider the cells B1,B2,C2,A3,A4 and then get its corresponding numerical tuple with the meaning: "Compared from the previous year, the change of inventory in current assets for 2019 is **18.1%**.". Therefore, extracting numerical tuples from tables is a vital while challenging task. Broadly speaking, the process of extracting numerical tuples can be imaged as an inverse process of table making. Before users create a table, the high-dimensional raw data exists in their minds as independent numerical tuples. These independent high-dimensional raw data are then partitioned, grouped, and arranged to form a highly coupled table based on their correlations. Conversely, extracting numerical tuples is the process of decoupling a complicated table into several independent numerical tuples. Therefore, the numerical tuple can be considered a basic form of data stored in tables.

Understanding the contents in cells and the relationships between them is crucial for extracting numeric tuples. The relationships between cells can be broadly classified into *Hierarchy* and *Juxtaposition*. (1) *Hierarchy*. The hierarchical structures in tables mainly indicate the meaning of belonging and dependency. It depends on their textual semantics and visual appearances. For example, in Figure 2(a), the differences of font style and indentation between A3 and A4, A5 indicate their hierarchical relationships. The relationships can also be inferred through their texts with some

	A	B	C
1		2019	
2		Assets	Changes from the Previous Year (%)
3	**Current**	21,614	12.4
4	Inventories	16,883	18.1
5	Cash and cash equivalents	4,731	-4.2
	…	…	…

B1 A3 B2 B3 A4

In 2019, the current assets amounted to 21,614, of which the inventories amounted to 16,883, cash and cash equivalents amounted to 4,731. They changed from the previous yeas is 12.4%, 18.1%, and -4.2%, respectively.

Figure 3: The difference between tables and natural languages. The content words in blue boxes are placed in table cells while the function words in red transparent boxes disappear. Best viewed in color.

finance knowledge (e.g., "Current Assets contain Inventories"). (2) *Juxtaposition*. The juxtaposition relationships between cells may express a semantic continuity. Take Figure 2(a) as a simple example. The cell B2 and C2 are juxtaposed since they are sibling nodes in the hierarchy tree. Although there is no hierarchical relationship between them, the meaning of C2 is carried over from B2, i.e., the meaning of C2 is "Assets' changes from the previous year (%)". A more complex example is shown in Figure 2(b), which we explain in Section 2.2. The semantic continuity between juxtaposition cells is implicitly expressed in the textual semantics of cells. In summary, extracting numerical tuples from tables requires deep modeling of textual semantics and visual appearances.

Traditional studies [4, 20] attempt to automatically extract relational tuples by first inferring the hierarchical tree of table headers and then constructing a numerical tuple with nodes on the path from each data cell to the tree root. Recent studies try to reduce the labor of users while extracting data in tables, such as integrating spreadsheet data with low human effort [5] or transforming spreadsheet data using some examples provided by users [1, 16]. Although these studies have made significant progress in extracting knowledge from tables, most of these methods confront three limitations. (1) They only extract tuples with hierarchical relationships but do not consider the juxtaposition between cells. (2) Some of them require algorithm-human interaction or rule sets made by domain experts, thus limiting the applicability of these methods. (3)

They only evaluate their systems on small corpora, such as SAUS R200 [5], WEB R200 [5], WEB R100 [4], TROY200 [20], which have up to 200 tables. The rule-based and machine learning methods on a small dataset are susceptible to overfitting or becoming unstable due to outliers.

In this paper, we propose a new framework to extract numerical tuples from tables and evaluate our framework on a large test set. Specifically, we suppose the value in a numerical tuple as the pivot and partition the tuple into several <*value, description*> pairs. Thus, the tuple extraction task is converted into a binary relation extraction task. Then we encode each cell into a hidden vector by a table representation model and aggregate vectors in each candidate pair to obtain their predicting result. Finally, the pairs with positive predictions are grouped into numerical tuples by values.

Obviously, representing a cell is the most important part of our framework. Motivated by the great success of the pre-trained language models on numerous tasks of natural languages [8] and images [9], a range of previous works have explored the potential of pre-trained models on tabular data. Existing BERT-like models usually flatten the table into a sequence treating it like a natural language [7, 14, 22, 25]. The difference between tables and natural languages is that the flattened sequence of the table usually contains only content words and lacks function words, as shown in Figure 3. Most function words are replaced by visual appearances in tables [24]. They express the critical grammatical relationships among texts while neglected in existing studies. To bridge the gap between table representation models and language models, we propose a BERT-based pre-trained language model, **TableLM**, to encode tables with diverse layouts. We encode the visual appearances as visual tokens added in the flattened sequence of tables. TableLM is aware of hierarchy and juxtaposition between cells by capturing textual semantics of cells and intricate correlations between texts and visual appearances. Compared to previous studies, our TableLM has four characteristics. (1) Our TableLM model can work on arbitrary types of tables because we do not make any assumptions in table layouts and contents. (2) We deploy a CNN to encode visual information, which is more reasonable to capture the commonality over various visual appearances. (3) We remove the numerical cell in the flattened sequences of tables because numerical cells do not provide much semantic meaning while are high in proportion and may introduce noise instead. (4) We pre-train the model on a large corpus based on contrastive learning between the contextual and local consistency of the cell. In Section 2, we explain the motivation of these designs in detail.

The evaluation on a large test set shows our framework achieves the F1-score of 85.63% at the tuple level, which outperforms a well-designed baseline (80.43%). Besides, we select more complicated tables from the test set to form a more challenging test set. The convincing results of experiments on the two test sets demonstrate the superiority of our model. We also conduct an ablation analysis to demonstrate the effectiveness of several model components.

2 PROBLEM DEFINITION AND CHALLENGES

2.1 Definition

In this section, we introduce the definition of the table, the numerical tuple, and the task used in this paper.

*Definition 2.1 (**Table**).* A table T can be expressed as a matrix with N rows and M columns: $T = \{c_{i,j} | 1 \le i \le N, 1 \le j \le M\}$, in which $c_{i,j}$ indicate the cell in the i-th row and j-th column. We define the set of cells that only contain values as numerical cell set T_v, and define non-numerical cell set as T_s, so $T = T_s \cup T_v$ and $T_s \cap T_v = \varnothing$ hold.

*Definition 2.2 (**Numerical Tuple**).* A numerical tuple in a table can be presented with the pair of $r = (v, D)$, which v is the pivot value of the numerical tuple, $D = \{d_i | 1 \le i \le K\}$ is the description set, K is the number of descriptions, and it may be different over tuples. Generally, v and the elements in D are a cell in table T, i.e., $v \in T$ and $d_i \in T$. We define R as all numerical tuples in a table.

*Definition 2.3 (**Numerical Tuple Extraction**).* Given a table T with its numerical cell set T_v and non-numerical cell set T_s, for each candidate cell $v' \in T_v$, our framework aims to extract its corresponding description set $D' \subset T_s$ to construct a tuple $r' = (v', D')$ that can express the full semantics of that cell. Note that the D' of some candidate cell v' can be empty, which means this is not a meaningful numerical tuple. The final predicted result of the framework is a set of numerical tuples: $R' = \{r'_1, r'_2, ...\}$.

2.2 Challenges

The challenges of extracting numerical tuples are summarized into two aspects.

2.2.1 The intricate correlations between cells. Elements of a numerical tuple are scattered in table cells and connected with their correlations. These correlations are expressed in the hierarchy and juxtaposition continuity of cells.

• Hierarchy. Hierarchy in tables can usually be expressed as trees, just like Figure 2(a). In general, the semantics of a node is constrained by its ancestor nodes. For example, in Figure 2(a), the "inventories" in cell A4 and A8 belong to "Current" and "Non-current" "Assets", respectively. The textual semantics and differences in visual appearances between cells express the hierarchy between cells.

• Juxtaposition. Two nodes with a juxtaposition may have a semantically progressive relationship. We explain the more complicated example in Figure 2(b). In this table, cell B1, C1, D1, E1, F1 are juxtaposed. However, there are complex semantic relationships between them that forms a directed graph as shown in the top of Figure 2(b), e.g., the meaning of F1 is carried over from B1 and E1, the meaning of D1 is carried over from B1 and C1. In general, the semantic continuity between juxtaposition cells mainly depends on their textual information.

Unfortunately, it is difficult for machines to understand the text or visual information, let alone capture such relations.

2.2.2 The textual and visual information of tables. Table is a two-dimensional multi-modal language. It adopts a different linguistic paradigm from natural language [24]. In tables, textual information carries semantic content, visual appearances chiefly express grammatical relationships, just as content words and function words in natural language.

• Textual semantics. Texts in cells are typically sentence fragments, leading the individual cell to hard express a complete meaning. Moreover, texts in tables may contain domain knowledge in specific domains. For example, the tables in Figure 2 consist of many

	A	B	C	D
1		2018		
2		Revenue	%	Changes from the Previous Year (%)
3	**Registered address**			
4	China	*****	*****	*****
5	Japan	*****	*****	*****
6	Singapore	*****	*****	*****
7	Korea	*****	*****	*****
8	Asia	*****	*****	*****
9	Rest of world	*****	*****	*****
10		*****	*****	*****

Figure 4: MLM loss is not ideally suitable for tables.

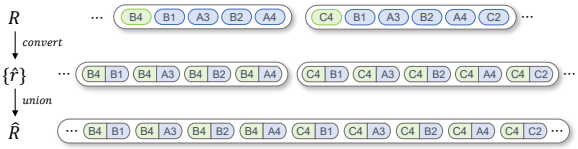


Figure 5: The third and fourth tuples in Figure 1 are converted by the process. We replace the text with their cell indexes. Best viewed in color.

proper terms from finance as the two tables come from companies' annual reports.

• The visual appearances. There are various visual appearances inside tables. For example, there are three kinds of visual appearances in the table of Figure 2(a). (1) Font styles. The bold font in cell A3, A6, A9 express hierarchical relationships between them and other cells with normal font in column A. (2) Indentations. The different level of indentations between cell A3, A6, A9 and other cells with no indentation in column A. (3) Separation lines. There are two horizontal separation lines below row 2 and 8, which divide the table into three parts. The first line separates the table header and data cells. The second line indicates the hierarchical level between row 9 and other rows.

Various visual appearances in tables are grouped in a coordinated and regular manner, expressing the relationship and semantic differences between cells. They express grammatical relationships like function words in natural language. However, visual appearances implicitly exist in tables and are hard to encode, unlike function words with some fixed tokens representing limited meanings.

2.3 The Motivation of Model Design

The proposed model TableLM is a BERT-based model with some specific designs motivated by several observations on tables.

Observation 1. *Numerical cells contain less semantics.* Typically, numerical cells contain only a specific value, which is not important for extracting their numerical tuples. There are fewer visual appearances and semantics in them. Even if we replace the value in a numerical cell with any other value, the descriptions of its numerical tuple do not change. Moreover, they are high in proportion and may introduce noise instead. In our dataset from finance, the average percentage of numerical cells in each table is 63.29%, leading to excessive tokens in the input sequence. Furthermore, it significantly increases memory and time consumption of BERT-based models, which is typically limited to the token length of 512 since the attention mechanism.

Motivated by this observation, we disregard the numerical cells in tables during encoding tables. Specifically, we delete all tokens in numerical cells in the flattened sequence of tables. Then, the representation of a numerical cell is obtained according to the non-numerical cells in the same row and column as it. This design significantly improves the efficiency of our model and enables the model to accommodate larger-sized tables.

Observation 2. *The commonality of visual appearances of tables.* There are many kinds of visual appearances in tables with flexible functionalities. Traditional studies [10, 13, 20] pay more attention to specific visual appearances and build rule-based feature systems. In contrast to them, we consider that various heterogeneous visual appearances are intended to express the **visual differences** between cells, which are homogeneous across tables. For example, in Figure 2(a), the different font styles and indentations between rows are designed to show that these rows differ in the hierarchy. The separation lines divide cells into different spaces, showing the difference between the cells. In addition, the same visual appearances hold among cells at the same level (e.g., row 4 and row 5 in Figure 2(a)). In summary, the different visual appearances or their combinations serve to express different hierarchies. Cells in the same hierarchical level have the same visual appearances. Therefore, we deploy a convolutional neural network to capture such visual differences over cells.

Observation 3. *BERT's masked-language modeling (MLM) loss is not ideally suitable for tables.* Masked language modeling loss is introduced from BERT [8]. It masks a part of random tokens in the input sequence and recovers them with other tokens by the model. However, tables often exemplify a series of entities with the same type, confusing the MLM loss target when recovering the tokens of masked entities. In Figure 4, if we mask the token "China" in cell A4 and want to recover it, the model may be confused between the target "China" and other country names. Such entity lists are widely found in tables but are rare in natural language. Motivated by the observation, we propose a new pre-training target based on contrastive learning. The idea is to require the model to learn high-level information such as entity type rather than recover the specific words. For example, in Figure 4, when the token "China" of cell A4 is masked, we require the model to learn the cell A4 should be a country instead restore the original "China".

3 FRAMEWORK

This section introduces our framework in detail. First, Section 3.1 introduces how to convert the problem of numerical tuples extraction into a binary relation extraction task. Then, the architecture of our TableLM and the procedure of pre-training and fine-tuning are described in Section 3.2, Section 3.3, and Section 3.4, respectively.

3.1 Framework Overview

The first question is how to get the numerical cell set T_v and non-numerical cell set T_s of a table. The numerical set T_v includes the cells that only contain values. It can be easily obtained by writing a simple regular expression script with some rules. Then we further

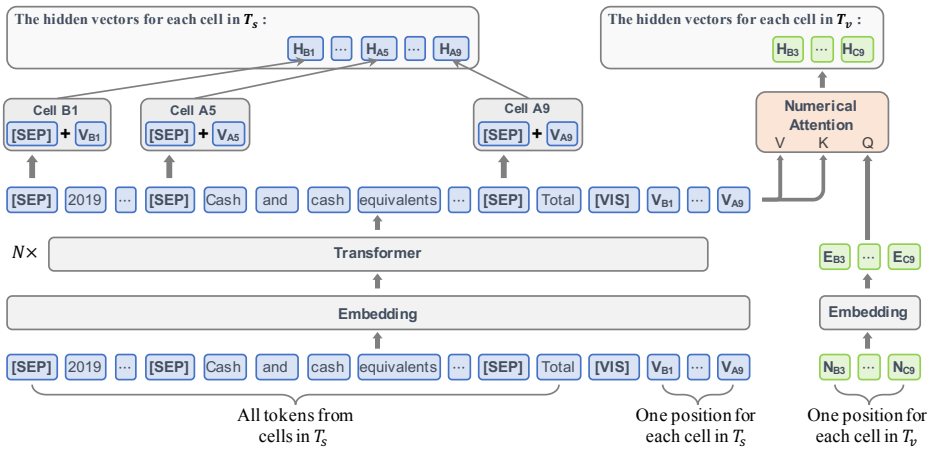


Figure 6: The architecture of TableLM. It encodes a table by capturing both textual and visual information. Best viewed in color.

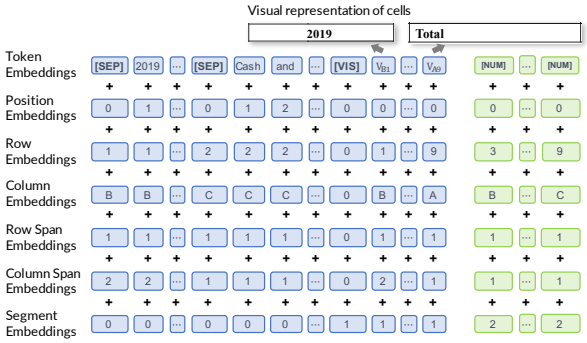


Figure 7: The embedding layer of TableLM.

get the non-numerical cell set: $T_s = T - T_v$. The statistics show that our script loses just **0.3%** correct numerical tuples in our dataset.

For each candidate value v' in T_v, we extract a subset D' from T_s, as the descriptions of the value. A numerical tuple can be presented as $r = (v, D)$. We split it into several pairs of $\hat{r} = \{(v, d_i)|d_i \in D\}$, Thus, the output becomes a union set of such pairs. Figure 5 illustrates the process. The original numerical tuples R are converted into a set of $\hat{r}$ and then are unified to the set of pairs $\hat{R}$.

Given a table T with its annotated numerical tuples R, we describe our framework in the training phase as follows.

(1) The annotated numerical tuples R are converted into $\hat{R}$.
(2) The two sets T_v and T_s are generated from T.
(3) We construct the candidate set of pairs: $R_c = \{(v', d')|v' \in T_v, d' \in T_s\}$. The label of each candidate pair is 1 if it belongs to $\hat{R}$ else 0.
(4) For each candidate pair, the TableLM model predicts its label.
(5) The parameters of TableLM model are updated according to the differences between predicted result and labels.

During the inference and testing phase, we can group pairs predicted to be positive by their values and obtain the final extracted numerical tuples.

3.2 TableLM Architecture

The architecture of TableLM (Figure 6) is based on BERT's encoders [8] with some extensions to encode visual appearances and tabular structures of tables. Generally, we treat the cells in the T_v and T_s in different ways. We only leverage the textual tokens and visual appearances of cells in T_s as the input. For numerical cells in T_v, we deploy a *Numerical Attention* module to obtain the representations of them based on the representations of cells in T_s.

To encode the visual appearances, we deploy the vision module of TaFor [24] ahead of the TableLM. This module first applies a CNN in pixels of the entire table and then another CNN at the cell level. It finally aggregates features into cells, allowing us to get a visual representation for each cell (including merged cells) by indexing. Then, we place the visual token of each cell in T_s at the end of the input sequence to complement the missing part of the flattened sequence as a language.

Next, we introduce: (1) how the embedding layer converts table content to input embeddings; (2) the detailed structure of N stacked Transformer encoders with our tabular attention; (3) how to get the hidden vector of each cell in a table.

3.2.1 *Embedding Layer.*

As shown in Figure 7, a table is flattened into a sequence of tokens which consists of three components: text part, vision part, numerical cell part. Note that although the input of the Transformer encoder does not contain numerical cells, the embedding layer also embeds each of them into an embedding vector. For embedding this sequence with its tabular structure, we extend the original embedding layer of the Transformer with several additional embeddings.

• **Token Embeddings.** For the text part, we concatenate all tokens in the non-numerical cells (T_s) and place a special token `[SEP]` at the beginning of each cell. For the vision part, the first is a special token `[VIS]` to separate the text part and vision part. The following are the visual representations of non-numerical cells in T_s. For the numerical cell part, we replace its value with a special token `[NUM]`, since the specific value in numerical cells is not important in our task. Finally, the tokens in the text and numerical cell parts

are converted to embedding vectors by an embedding table, the dimension of which is the same as visual representations.

• **Position Embeddings.** We take the index of each token in the cell instead of in the sequence as its position ID. Because the tokens in adjacent cells in tables are typically not semantically contiguous.

• **Row / Column Embeddings and Span Embeddings.** The tabular position of a cell can be presented by $(id_r, id_c, span_r, span_c)$, where id_r and id_c indicate the row index and column index of the cell, $span_r$ and $span_c$ indicate the number of rows and columns spanned by the cell. For example, the position of cell B1 is presented $(1, B, 1, 2)$. We add four embedding layers to encode them.

• **Segment Embeddings.** It takes two values: 0 for textual tokens and 1 for visual representations of cells. A final embedding layer to convert it into vector space.

3.2.2 *Transformer with Tabular Masked Attention.*

The Transformer has been widely used in natural language models and is well adapted for sequential data. A Transformer block is composed of N stacked multi-head self-attention layer followed by a point-wise, fully connected layer [8]. We perform a masked attention mechanism in the self-attention layer in the Transformer to make the model aware of the tabular data. We restrict each token in the self-attention layer to see only tokens in the same row and column as itself by using a symmetrical binary matrix M. Formally, the masked attention is design as follows:

$$\text{attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}}M)V. \tag{1}$$

Here Q, K, V are inputs of Attention function; d is the dimension of Q, K, V; M is the tabular visibility matrix. We define the tabular visibility matrix M to enable tabular attention, i.e., we set $M_{i,j} = 1$ if token_i and token_j are in the same row or column. We assume that a merged cell belongs to all the rows and columns it spans. Note that even though we restrict each token to interact with other tokens in the same row or column, they can still contact tokens that are not visible in matrix M. Because the receptive field of each token increases as the Transformer layer gets deeper.

3.2.3 *Cell Representations.*

After the stacked Transformer blocks, the output contains the hidden vectors of tokens in non-numerical cells and visual tokens. These tokens have interacted with each other. We first introduce how to get the hidden vector of each cell. Cells in tables are divided into two sets: non-numerical cells (T_s) and numerical cells (T_v).

• **Non-numerical cells (cells in T_s).** As shown in top left of Figure 6, the hidden vector of each non-numerical cell is calculated with the output hidden of its special token `[SEP]` and its visual token $V_{i,j}$:

$$H_{i,j} = \text{LayerNorm}([\text{SEP}]_{i,j} + V_{i,j}) \tag{2}$$

• **Numerical cells (cell in T_v).** We perform *numerical attention* to obtain the hidden vector of numerical cells, shown in the top right of Figure 6. The structure of numerical attention is the same as multi-head attention in the Transformer, except that the Q of attention is the embeddings of numerical cells, and the K, V of attention are the output hidden of the Transformer. We also perform the tabular visibility matrix M in the numerical attention.

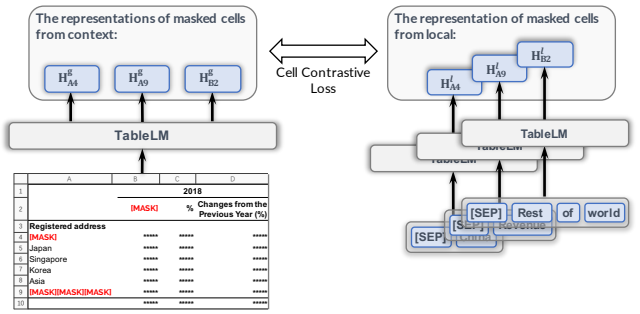


Figure 8: Illustrations for Cell Contrastive Loss. The original table is in Figure 4

3.3 Pre-training TableLM

We only pre-train the stacked Transformer that does not involve numerical cells, so they are not considered during pre-training.

As we mentioned in Section 2.3, the MLM loss in BERT is not ideally suitable for tables. Therefore, we perform a new pre-training objective to learn the high-level information of each cell with contrastive learning. In addition, we also adopt the Masked Language Modeling objective with the whole-cell masking strategy to capture the relationships between cells as [22]. The pre-training objectives are introduced as follows.

• **Cell Contrastive Loss (CCL).** Cells are the basic unit of the table to list data and record text segments. Moreover, the elements of numerical tuples are cells. Therefore, the representations of cells are crucial for our task of numerical tuple extraction. Cell Contrastive Loss learns high-level information about cells through the supervision of intermediate representations. The procedure of calculating CCL is illustrated in Figure 8. We first randomly select cells with a probability of 15%, then replace each of their textual tokens with a special token `[MASK]`. On the one hand, we feed the sequence with masked tokens into TableLM, outputting the hidden representations of masked cells generated by their context, e.g., $H^g_{A4}, H^g_{A9}, H^g_{B2}$ in the top left of Figure 8. On the other hand, for each masked cell, we construct a sequence consisting of all the original words in it and a special leading token `[SEP]` and feed it into TableLM. It can be interpreted that we input each masked cell into TableLM as a separate table. In this way, the hidden representations of masked cells generated by their local text are obtained from their special leading token `[SEP]`, as shown in the top right of Figure 8. Note that these TableLMs in Figure 8 share their parameters. Then we perform the contrastive prediction task inspired by SimCLR [3]. For the contextual representation of a masked cell, we treat its local representation as its positive example and the local representations of other masked cells in the same minibatch as negative examples. Then, we maximize the similarity between positive pairs and minimize the similarity between negative pairs. More formally, let $\text{sim}(u, v) = u^T v/(\|u\| \, \|v\|)$ denote the cosine similarity between u and v, $\mathcal{M}$ denote the set of masked cells in the batch. The loss function of CCL is defined as:

$$\mathcal{L}_{\text{CCL}} = -\sum_{c_i \in \mathcal{M}} \log \frac{\exp(\text{sim}(H^g_{c_i}, H^l_{c_i})/\tau)}{\sum_{c_j \in \mathcal{M}} \exp(\text{sim}(H^g_{c_i}, H^l_{c_j})/\tau)}, \tag{3}$$

where τ denotes a temperature parameter.

• **Masked Language Modeling (MLM).** Masked language Modeling objective randomly masks tokens in the input sequence and predicts these masked tokens. In TableLM, we adopt the MLM objective upon the selected masked tokens in the CCL objective. Let $\mathcal{L}_{MLM}$ denote the loss of MLM objective.

The final pre-training loss is calculated as $\mathcal{L} = \mathcal{L}_{CCL} + \mathcal{L}_{MLM}$.

3.4 Fine-tuning TableLM on Numerical Tuple Extraction

With pre-training, TableLM learns the knowledge about various types of tables from a large table dataset. We leverage the parameter of the stacked Transformer as the initial parameter during fine-tuning. For the task of numerical tuple extraction, we need to calculate the prediction for each candidate pair of numerical cell and non-numerical cell (v', d'). We concatenate their hidden representations and feed it into a FFN [21], followed by a softmax layer for binary classification. The cross-entropy loss is defined as:

$$p' = \text{softmax}(\text{FFN}(\text{concat}(H_{v'}, H_{d'}))), \tag{4}$$

$$loss = -\sum_{c=0}^{1} y_c \log\left(p'_c\right) + (1 - y_c) \log\left(1 - p'_c\right), \tag{5}$$

where FFN consists of two linear transformations with a ReLU activation in between, p' is the predicted distribution, p'_c is the probability that the candidate pair is classified into class c.

4 EXPERIMENTS

4.1 Dataset

To evaluate our framework on a large corpus, we collected a dataset consisting of 19,264 tables from Chinese financial documents crawled from CNINFO[1], a website for the China Securities Regulatory Commission. We call the dataset **FinTab-Tuples**. The tables in finance are data-intensive containing numerous numerical tuples. Because recognizing numerical tuples requires extensive domain knowledge, we recruited nine in-house annotators who have been working in the finance industry for more than three years. We trained them to construct the dataset. For each numerical cell in a table, the first two annotators are responsible for independently annotating the cells that have a semantic relationship with the numerical cell. If these two annotators have conflicting results, another annotator proofreads and corrects the result until there is no disagreement between the three annotators. The resultant dataset contains 19,264 tables with a total of 604K labeled numerical tuples. We divide the dataset into training, validation, and test sets in the ratio of 8:1:1.

To demonstrate the difficulty of our dataset, we define complex tuples as follows: if a tuple contains a description that is not in the same row or column as the value of the tuple, we call it a complex tuple. Here, if a cell spans multiple rows/columns, we consider it to be in the same row/column as the cells in the rows/columns it spans. Furthermore, we define the table containing at least one complex tuple as a complex table. Table 1 gives some statistics about our dataset. There are more than 600K labeled numerical tuples and 191K complex tuples in our dataset. We consider a numerical tuple to correspond to its value cell, and more than half of cells (58.19%) per table contain tuples. It demonstrates that the tables in our dataset are data-intensive. The average proportions of complex tuples in tuples per table and per complex table are 27.22% and 58.90%, respectively. The average size of tables is not too large (average 9.32 rows and 5.88 columns) because the tables are designed on the page of a financial document for human reading. In addition, a table with dozens or hundreds of rows is usually similar to a relational table, used to list a series of records. These tables are in turn relatively easy to analyze for machines.

[1]http://www.cninfo.com.cn/

Table 1: Statistic of FinTab-Tuples

# tables	19,264
# complex tables	8,906
# labeled tuples	604,111
# labeled complex tuples	191,344
Avg. % numerical cells per table	63.29%
Avg. % tuples in cells per table	58.19%
Avg. % complex tuples in tuples per table	27.22%
Avg. % tuples in cells per complex table	60.01%
Avg. % complex tuples in tuples per complex table	58.90%
Avg. # rows per table	9.32
Avg. # columns per table	5.88

The test set of our dataset, FinTab-Tuples-T, consists of 1,927 tables, which is around ten times large than the datasets used in previous studies [4, 5, 20]. To evaluate the effectiveness of our framework on complicated tables, we collect all complicated tables in the test set of FinTab-Tuples to form a more challenging test set **FinTab-Tuples-CT**. The experiments on the two test sets produce more convincing results and analyses.

4.2 Pre-training Details

4.2.1 Dataset for Pre-training. In order to boost the performance of the TableLM, we pre-train it on the table dataset FinFormulas [24]. The FinFormulas dataset consists of 190,179 tables from various types of 4,746 Chinese financial documents, also crawled from CNINFO. We ensured that the two datasets FinTab-Tuple and FinFormulas do not contain duplicate tables to avoid the problem of data leakage.

4.2.2 Pre-training Configuration. TableLM is based on BERT architecture, so its configuration aligns with $\text{BERT}_{\text{BASE}}$. Since we adopt in-cell position embedding to capture the semantics of text segments in cells, the parameters of TableLM are initialized with BERT's token embeddings, position embeddings, and encoder weights. Specifically, we start pre-training from Chinese-BERT with whole word masking [6]. We pre-train TableLM for 100 epochs with 10K warm-up steps on 7 GeForce RTX 2080 Ti. The pre-training procedure takes around 7 days with a batch size of 4 tables in each GPU. The temperature in Cell Contrastive Loss is set to 0.07. AdamW [18] and linearly decayed learning rate schedule with an initial 5e-5 are deployed for pre-training. We keep the first 20 tokens in each cell and filter out tables whose flattened sequences are longer than 800. Since we use in-cell position rather than global position in the input sequence, the input length of our model can exceed 512.

4.3 Experiments Setup

4.3.1 Fine-tuning Configuration. In the fine-tuning procedure, we initialize the parameters of the stacked Transformer in TableLM from the pre-training result. The parameters of numerical attention are initialized randomly. The hidden size in numerical attention is the same as the multi-head attention in Transformer of TableLM, which is 768. The model is trained for 200 epochs with a batch size of 15 and a learning rate of 3e-5 in the training set of FinTab-Tuples.

4.3.2 Metric. We have three levels of metrics to demonstrate the effectiveness of our framework.

(1) F1-score at pair level. It can be easily calculated by comparing candidate pairs' predictions and ground truth labels.

(2) F1-score at tuple level. We can define the precision and recall at tuple level as:

$$p = \frac{|R' \cap R|}{|R'|}, r = \frac{|R' \cap R|}{|R|}. \tag{6}$$

Here, R denotes the set of ground truth numerical tuples, R' denotes the set of predicted results, $|\cdot|$ indicates the length of a set. Then, the F1-score is defined as: $F1 = (2 * p * r)/(p + r)$.

(3) Table Level Accuracy. We calculate the percentage of tables whose numerical tuples are all correctly extracted.

4.3.3 Compared Methods. We compare our TableLM against with TaFor [24] based on deep neural networks. TaFor is proposed to represent a table for recognizing its formulas. It has the same functionality as TableLM, which encodes a table and produces hidden representations of its cells. TaFor leverages a CNN to capture the visual information of tables and then performs two LSTM to capture the correlation within row headers and column headers, respectively. Finally, the hidden vector of each cell is obtained by its corresponding row header and column header. In the comparison experiment of TaFor, we adopt the same framework introduced in this paper, except that replacing TableLM with TaFor. We use the same configuration as their paper.

4.4 Effectiveness Evaluation

4.4.1 Method Performances. Table 2 lists the results of our TableLM compared with the baseline TaFor. The results on the test set FinTab-Tuple-T shows that the proposed TableLM achieves superior performance with 71.44% in table accuracy, $F1 = 96.99\%$ at pair level, and $F1 = 85.63\%$ at tuple level, which are higher than TaFor (63.06%, 95.53%, 80.43%, respectively). When evaluated on the challenging test set FinTab-Tuples-CT, the performance of all methods degrades to some extent, as shown in the last three columns in Table 2. Such a decline illustrates that complex tuples in the dataset are more challenging to extract. In this case, our TableLM still outperforms the baseline of TaFor (79.44% vs 74.28% at tuple level). Note that the CNN for capturing visual information in TableLM is borrowed from TaFor. It indicates that the Transformer structure is better at capturing the correlations between tokens and visual appearances in tables than LSTM.

4.4.2 Ablation Study. In Table 3, we conduct an ablation study to evaluate the effectiveness of components of our model.

• The effectiveness of visual information. Our TableLM encodes visual appearances as visual tokens and places them into stacked Transformer to complement the missing part of texts in tables as a language. When we eliminate the visual tokens in TableLM, its performance drops dramatically from 85.63% to 77.17% (↓8.46%) at the tuple level. The experiments on the challenging set FinTab-Tuples-CT obtain consistent results. It demonstrates the importance of modeling visual appearances in table representation models.

• The effectiveness of the CCL objective. When we remove the CCL objective, the performance on FinTab-Tuples-T decreases by table accuracy with 4.95% (71.44% → 66.49%) and tuple level F1-score with 2.09% (85.63% → 83.54%). When conducting the ablation on FinTab-Tuples-CT, the F1-score at pair and tuple levels is still higher. Clearly, the CCL helps our TableLM achieve better results.

• The effectiveness of pre-training. We train the TableLM from scratch and report the results on Table 3. On the test set FinTab-Tuples-T, its performance at tuple level is lower than full TableLM with around 2% and on par with TableLM without CCL. However, on the test set FinTab-Tuples-CT, the performance of TableLM without pre-training drops significantly. Since the test set FinTab-Tuples-CT contains hundreds of complicated tables, it demonstrates that the pre-training improves the model's ability to capture the semantics of complicated tables.

Table 2: Results (%) of methods on two test sets. Here, Acc. is an abbreviation for accuracy, F1-P is the F1-score at pair level, F1-T is the F1-score at tuple level.

	FinTab-Tuples-T			FinTab-Tuples-CT		
	Acc.	F1-P	F1-T	Acc.	F1-P	F1-T
TaFor	63.06	95.53	80.43	56.47	94.91	74.28
TableLM	**71.44**	**96.99**	**85.63**	**63.58**	**96.20**	**79.44**

Table 3: Ablation Results (%) on two test sets. Here Acc., F1-P, F1-T are the same as Table 2.

	FinTab-Tuples-T			FinTab-Tuples-CT		
	Acc.	F1-P	F1-T	Acc.	F1-P	F1-T
TableLM	**71.44**	**96.99**	**85.63**	63.58	**96.20**	**79.44**
w/o vision	54.34	95.25	77.17	55.53	94.30	71.52
w/o CCL	66.49	96.51	83.54	**64.58**	95.66	78.34
from scratch	63.47	96.58	83.66	58.66	94.84	74.98

5 RELATED WORK

Data Extraction in Tables. Early studies transform data in a relational table into databases by mapping the attributes of original tables and target databases [12, 19]. Hung et al. [15], Shigarov and Mikhailov [20] proposed a rule-based method that requires conversion rules provided by users. These methods assume the input table is relational and are failed to handle other types of tables. The hierarchical relationships between cells were observed by Chen and Cafarella [4], Shigarov and Mikhailov [20]. Chen and Cafarella [4] first inferred the hierarchical tree of table headers and then constructed a relational tuple with nodes on the path from each data cell to the tree root. Shigarov and Mikhailov [20] proposed a rule-based tool TabbyXL based on a domain-specific language for expressing table analysis and interpretation rules. They analyze many types of table hierarchical layouts. However, the two

methods neglect juxtaposition relationships among cells and only evaluated their methods on the performance of inferring the hierarchical structure of tables. Chen and Cafarella [5] leveraged the user feedback for extracting hierarchy structure in their semiautomatic method. Barowy et al. [1], Jin et al. [16] perform data transformation based on the examples provided by users. These methods require a human-computer interaction process.

Compared with existing works, the proposed framework leverages deep neural networks' powerful representational learning capabilities. It obtains the representation of cells end-to-end, avoiding massive investment in table layout and tedious rules made by domain experts. Moreover, we evaluate our model on a large corpus around ten times larger than the dataset in previous works.

Table Representations. In fact, a range of prior work has explored the potential of pre-trained Transformers on tabular data. TAPAS [14] and TaBERT [25] from the NLP community pre-trained Transformers over tables and texts to address question answering on relational tables. TURL [7] proposed a structure-aware Transformer encoder with a new Masked Entity Recovery objective to capture knowledge embedded in the relational Web tables. Their structure-aware Transformer inspires our tabular masked attention. However, compared with them, our TableLM can be adopted in arbitrary tables since we do not make any assumption in table layouts. TUTA [22] define two bi-dimensional coordinate trees of row headers and column headers and encode them in Transformers. They pre-trained their model jointly with two new objectives with the MLM objective. Interestingly, the Cell-level Cloze in TUTA is also designed to learn representations at the cell level, similar to our Cell Contrastive Loss. However, our Cell Contrastive Loss pays more attention to cells' high-level semantics (e.g., entity types). Some of them encode visual appearances with several hand-crafted features [10, 13, 20], which is different from our visual tokens. It is more reasonable that use CNN to capture the commonality among visual appearances. A more similar design appears in LayoutLMv2 [23], where the authors split a document page into four parts and create one visual token for each parts using a CNN-based visual encoder.

Existing BERT-based methods focus more on the table's semantic and spatial structure information. TaFor also learns the representations of cells in tables by modeling both textual semantics and visual appearances of tables. We borrowed from TaFor to adopt a CNN to encode the visual appearances of a table.

6 CONCLUSION

In this paper, we explore the task of extracting numerical tuples from tables with the pre-training language models, TableLM. We propose a framework based on the TableLM which converts the task into a binary classification of cell pairs. The TableLM is a BERT-based language model for tabular data. It consider both texts and visual appearances as the model's input tokens. Finally, we propose a new objective Cell Contrastive Loss to pre-train the TableLM jointly with the MLM objective. In future work, we will continually explore the application of extracted numerical tuples to various downstream tasks to evaluate the effects of this task. In addition, we will conduct more experiments on various tasks of table understanding to demonstrate the effectiveness of our pre-training model TableLM.

ACKNOWLEDGMENTS

The research work was supported by the National Natural Science Foundation of China under Grant No. 62076231 and U1811461.

REFERENCES

[1] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. 2015. FlashRelate: extracting relational data from semi-structured spreadsheets using examples. *ACM SIGPLAN Notices* (2015), 218–228.
[2] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment* 1, 1 (2008), 538–549.
[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *ICML*. 1597–1607.
[4] Zhe Chen and Michael Cafarella. 2013. Automatic web spreadsheet data extraction. In *International Workshop on Semantic Search over the Web*.
[5] Zhe Chen and Michael Cafarella. 2014. Integrating spreadsheet data via accurate and low-effort extraction. In *KDD*.
[6] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting Pre-Trained Models for Chinese Natural Language Processing. In *ENMLP*. 657–668.
[7] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. In *VLDB*.
[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*. 4171–4186.
[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*.
[10] Lun Du, Fei Gao, Xu Chen, Ran Jia, Junshan Wang, Jiang Zhang, Shi Han, and Dongmei Zhang. 2021. TabularNet: A Neural Network Architecture for Understanding Semantic Structures of Tabular Data. In *KDD*.
[11] Mary Jo Foley. 2010. About that 1 billion Microsoft Office figure ... https://www.zdnet.com/article/about-that-1-billion-microsoft-office-figure/. Accessed June 16, 2010.
[12] Ariel Fuxman, Mauricio A. Hernández, C. T. Howard Ho, Renée J. Miller, Paolo Papotti, and Lucian Popa. 2006. Nested mappings: schema mapping reloaded. In *VLDB*.
[13] Majid Ghasemi-gol, Jay Pujara, and Pedro Szekely. 2019. Tabular Cell Classification Using Pre-Trained Cell Embeddings. In *ICDM*.
[14] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *ACL*. 4320–4333.
[15] Vu Hung, Boualem Benatallah, and Regis Saint-Paul. 2011. Spreadsheet-based complex data transformation. In *CIKM*. 1749–1754.
[16] Zhongjun Jin, Michael R. Anderson, Michael Cafarella, and H. V. Jagadish. 2017. Foofah: Transforming Data By Example. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. 683–698.
[17] Hongwei Li, Qingping Yang, Yixuan Cao, Jiaquan Yao, and Ping Luo. 2020. Cracking Tabular Presentation Diversity for Automatic Cross-Checking over Numerical Facts. In *KDD*.
[18] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *ICLR*.
[19] Alessandro Raffio, Daniele Braga, Stefano Ceri, Paolo Papotti, and Mauricio A Hernández. 2008. Clip: a visual language for explicit schema mappings. In *ICDE*. IEEE, 30–39.
[20] Alexey O Shigarov and Andrey A Mikhailov. 2017. Rule-based spreadsheet data transformation from arbitrary to relational tables. *Information Systems* 71 (2017), 123–136.
[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*.
[22] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. TUTA: Tree-based Transformers for Generally Structured Table Pre-training. In *KDD*. 1780–1790.
[23] Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, Min Zhang, and Lidong Zhou. 2021. LayoutLMv2: Multi-modal Pre-training for Visually-Rich Document Understanding. In *ACL*.
[24] Qingping Yang, Yixuan Cao, Hongwei Li, and Ping Luo. 2021. Numerical Formula Recognition from Tables. In *KDD*. 1986–1996.
[25] Pengcheng Yin, Graham Neubig, Wen-Tau Yih, and Sebastian Riedel. 2020. TABERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*.