Metric-Type Identification for Multi-Level Header Numerical Tables in Scientific Papers

Lya Hulliyyatus Suadaa[1], **Hidetaka Kamigaito**[1], **Manabu Okumura**[1], **Hiroya Takamura**[1,2]
[1]Tokyo Institute of Technology
[2]National Institute of Advanced Industrial Science and Technology (AIST)
lya@stis.ac.id
{kamigaito,oku}@lr.pi.titech.ac.jp
takamura@pi.titech.ac.jp

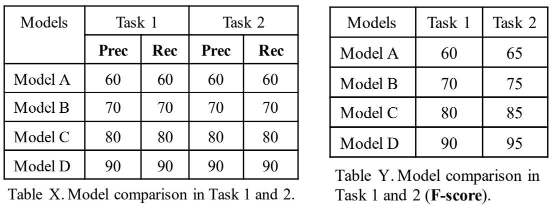
Abstract

Numerical tables are widely used to present experimental results in scientific papers. For table understanding, a metric-type is essential to discriminate numbers in the tables. We introduce a new information extraction task, metric-type identification from multi-level header numerical tables, and provide a dataset extracted from scientific papers consisting of header tables, captions, and metric-types. We then propose two joint-learning neural classification and generation schemes featuring pointer-generator-based and BERT-based models. Our results show that the joint models can handle both in-header and out-of-header metric-type identification problems.

1 Introduction

Tables are powerful tools for presenting data efficiently in row and column views. In scientific papers, numerical tables are commonly used to show experimental results for facilitating data analysis. Examples of numerical tables in scientific papers are shown in Figure 1.

Tables have the ability to cover multiple categories written in table headers by incorporating several header sets in a hierarchical view, called multi-level header tables. Scientific papers have strict guidelines about tables; for example, one states that a similar type of text is written in the same level of header. Figure 1a shows a multi-level header example in the column part, with task type (*Task 1* and *Task 2*) in the first header-level and metric-type (*Prec* and *Rec*) in the second. The table also has a row header specifying the model type (*Model A*, *Model B*, *Model C*, and *Model D*). In the real-world, this header-type information is limited due to the unknown table scheme. However, we assume tables in scientific papers follow the rule of categorizing a similar type of header name in the same header-level.



Models	Task 1		Task 2	
	Prec	**Rec**	**Prec**	**Rec**
Model A	60	60	60	60
Model B	70	70	70	70
Model C	80	80	80	80
Model D	90	90	90	90

Table X. Model comparison in Task 1 and 2.

Models	Task 1	Task 2
Model A	60	65
Model B	70	75
Model C	80	85
Model D	90	95

Table Y. Model comparison in Task 1 and 2 (**F-score**).

(a) Metric-type in header (b) No metric-type in header

Figure 1: Example tables in scientific papers. Bold indicates their metric-type.

To understand the numbers in the tables, metric-types are important for discriminating the numbers. A comparison between numbers is applied for numbers in the same metric-type with different categories. For the table in Figure 1a, we cannot compare the number 60 for *Model A* in the first column with 60 in the second one because they have a different metric-type: *Prec* and *Rec*. Computing numbers with different metric-types will result in inaccurate analysis.

Different tables may have different ways of writing their header name, such as using abbreviations like *p*, *pre*, or *prec* to refer to *precision*. Due to the lexical diversity of header names, metric-type identification becomes more challenging. Using a rule-based metric-type tagging or a limited set of metric-types in a dictionary is not enough to cover the diversity. Since tables in scientific papers typically have logical captions and logical categorization of the header-level, we introduce a metric-type identification task that locates the metric-type in the headers by using the caption and header name as inputs. For the example shown in Figure 1a, the metric-type is located in the second level of the column header.

We also cover tables that do not mention metric-types in their header (out-of-headers), as shown in Figure 1b. In these cases, the metric-types are identified in the caption. To cover metric-types located both in the headers and not in the headers, we propose a joint framework of metric-type location prediction and metric-type token generation for the metric-type identification task in multi-level header tables.

Our contributions are as follows:

- We introduce a metric-type identification task for multi-level header tables and propose joint location prediction and generation models to solve the task.
- We provide a dataset consisting of multi-level header numerical tables, captions, and metric-types, extracted from scientific papers. Our datasets will be publicly available[1].
- We introduce a multi-level header table encoder mechanism to obtain table header representations and propose a pointer-generator-based model to cover out-of-headers in the metric-type identification task.
- We fine-tune a general pre-trained encoder (BERT) and a domain-specific encoder (SciBERT) in our task and present the experimental results. We show that the models incorporating the pre-trained encoders lead to significant performance gains, especially when using a domain-specific one.

2 Related Work

Table information extraction is beneficial to cover unknown table schemes and understand the table contents. Milosevic et al. (2019) proposed a framework for table information extraction in biomedical domains by defining rules for all possible variables. Specifically, for numerical variables, they retrieved metric-types by searching a set of possible tokens in the dictionary. Focusing on numerical tables, Nourbakhsh et al. (2020) extracted metric-types in earning reports by using similarity scores between the corresponding non-numeric text for the leftmost cells and stored metric-types.

The work closest to ours is the one by Hou et al. (2019), who used tables from the experimental result section, combined with the title and abstract as document representations to extract triples of tasks, dataset, and metric for leaderboard construction. In our study, we represent the tables in more generic ways, preventing the original table structure in the multi-level headers form. We intend to retain the ability of a table to cover complex categorization in the headers and efficiently present all values. A previous study that also explored multi-dimensional tables was done by Milosevic et al. (2016) to automatically detect table structures from XML tables.

Our pointer-generator-based model in the metric-type generation scheme is inspired by the promising results of the pointer-generator network (See et al., 2017) in the summarization task. The network deals with the out-of-vocabulary issue by joint copying from source texts and generating from vocabularies.

Recent studies have shown that pre-trained encoders can be successfully fine-tuned for downstream NLP tasks, thus avoiding the need to train a new model from scratch. A pre-trained encoder BERT (Devlin et al., 2019) was trained on the BooksCorpus (800M words) and Wikipedia (2,500M words). For better-contextualized representation in the scientific domain, Beltagy et al. (2019) introduced a domain-specific BERT model, SciBERT, which was trained on 1.14M papers from Semantic Scholar. Friedrich et al. (2020) implemented both BERT and SciBERT on their models to solve the information extraction task and achieved significant performance gains.

3 Metric-Type Identification for Numerical Tables

3.1 Datasets

We automatically extracted tables from the PDF files of scientific papers in the computational linguistics domain using PDFMiner and Tabula as extraction tools and filtered only numerical tables related to experimental results using the keywords *evaluation*, *result*, *comparison*, and *performance*. We used papers from the ACL and EMNLP conferences (2016 to 2019) on the ACL Anthology website as data sources.

In tables in actual scientific papers, knowledge about the table semantics is rarely provided. On the basis of how information is "read" from a table, Hurst (2000) separated functional table areas into access cells and data cells. Access cells consist of column headers and/or row headers. We define data structure on the basis of their functional areas: table caption (capt), row headers (rh), column headers

[1]Dataset is available on https://github.com/titech-nlp/metrictable

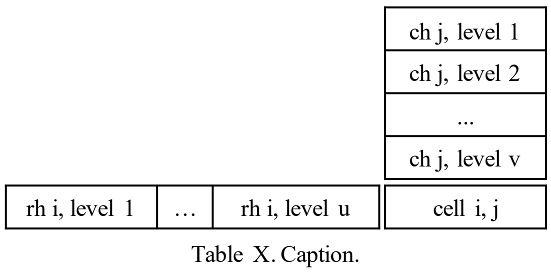
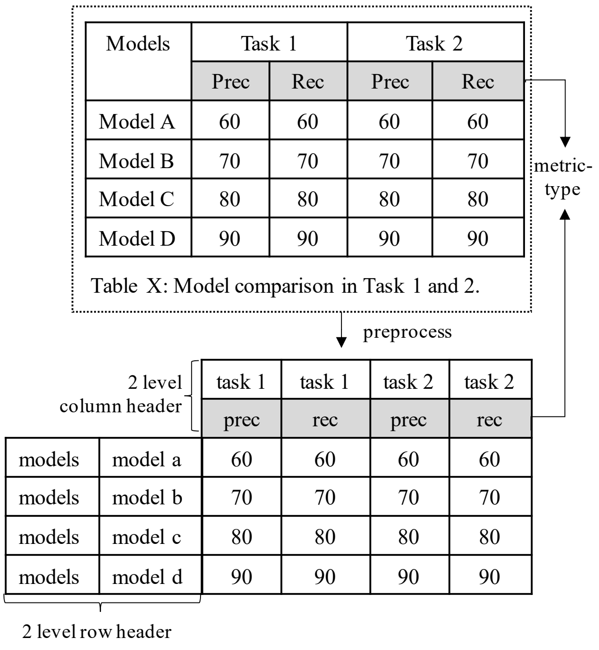


Figure 2: Table structure.



Models	Task 1		Task 2	
	Prec	Rec	Prec	Rec
Model A	60	60	60	60
Model B	70	70	70	70
Model C	80	80	80	80
Model D	90	90	90	90

Table X: Model comparison in Task 1 and 2.

		task 1	task 1	task 2	task 2
		prec	rec	prec	rec
models	model a	60	60	60	60
models	model b	70	70	70	70
models	model c	80	80	80	80
models	model d	90	90	90	90

Figure 3: Illustration of table preprocessing.

(ch), and cells. Headers in the row and column parts have several levels, and we assume that header names in the same level have the same type. Figure 2 shows our table structure.

We hired several qualified workers in the computer science field to manually check the extracted table structure to ensure the separation of row headers, column headers, and cells was correct, as shown in Figure 3. Then, they annotated the metric-type of the tables by prioritizing the locating of the metric-type in a specific header-level. The annotators were able to identify the metric-types of approximately 70% of the tables in their headers, and they determined the metric-type of the rest of the tables on the basis of information from the table captions. When no metric-type was mentioned in the headers, we assumed the metric-type was the same for all table values. The structures from the example in Figure 3 are capt: "*model comparison in task 1 and 2*"; rh level 1: [*models, models, models, models*]; rh level 2: [*model a, model b, model c, model d*]; ch level 1: [*task 1, task 1, task 2, task 2*]; ch level 2: [*prec, rec, prec, rec*]; and metric-type: [*prec, rec, prec, rec*] (identified in ch level 2).

We split our dataset into training, validation, and test sets. The statistics of our dataset are provided in Table 1.

	Train	**Val**	**Test**
No. of tables	1,084	136	135
Average row/column	6	6	5
Max level:			
- row header	9	6	4
- column header	6	5	6
Vocab size:			
- headers	8,270	1,435	1,230
- all metric-types	807	175	185
- unique metric-types	90	22	28

Table 1: Dataset statistics in training, validation, and test sets.

3.2 Problem Definition

Let Table $= \{capt, rh_k^i, ch_l^j, cell_{ij}\}$, where $1 \leq i \leq n_r, 1 \leq j \leq n_c, 1 \leq k \leq u, 1 \leq l \leq v$ denote an $n_r \times n_c$ table with the u level of rh and v level of ch. The task is to identify metric-type set ($\hat{m}$) in the specific level of row header (rh_k) and column header (ch_l). To handle tables that do not include metric-types in their headers, we generate $\hat{m}$ by using information from the table caption. The formulation of the metric-type identification is as follows:

$$\hat{m} = \begin{cases} \{rh_k^i\}_{i=1}^{n_r}, k \in \{1, ..., u\} & \text{if } \hat{m} \text{ in } rh \\ \{ch_l^j\}_{j=1}^{n_c}, l \in \{1, ..., v\} & \text{if } \hat{m} \text{ in } ch \\ \{w_m\}_{\times j}, w_m \in W_m \text{ or} & \\ \qquad\qquad w_m \in capt & \text{otherwise,} \end{cases} \tag{1}$$

where W_m is a set of metric-types in the vocabulary.

4 Models

We propose neural models to identify the metric-type for multi-level header tables by means of a joint model of metric-type location prediction and metric-type token generation.

4.1 Pointer-Generator with Supervised Attention Model

We obtain the representations of captions and header-levels by using a BiLSTM encoder and then capture header-level weights using supervised attention between the header-level encoder and the metric-type header-location outputs. In the generation scheme, we adopt the pointer-generator network to take into account captions as source texts and the metric-type vocabulary in the metric-type generation gate. The architecture of our model is shown in Figure 4.

Header encoder We use the vector representation of each header-level by averaging the vectors of all header name tokens in the same level. Given E_{rh_k} and E_{ch_l} as the averages of the initial vector representations of the row and column header-level vectors, respectively, we use the BiLSTM encoder with the dot attention mechanism proposed by Luong et al. (2015) to obtain the representations of the row and column header-levels and select the last hidden state of the last level combined with the weighted hidden states as header-level contexts, as follows:

$$C_{rh} = [C_{rh_u}; \sum_{k=1}^{u} a_{rh_k} C_{rh_k}], \tag{2}$$

$$C_{ch} = [C_{ch_v}; \sum_{l=1}^{v} a_{ch_l} C_{ch_l}]. \tag{3}$$

Caption encoder As with the headers, we use the BiLSTM encoder with attention a_{capt_i} to compute the context vector of caption C_{capt}.

Metric-type header-location gates We feed the concatenation of the row and column header contexts to the softmax layer to obtain the metric-type header-location probability:

$$p_{hloc} = \text{softmax}([C_{rh}; C_{ch}]), \tag{4}$$

which includes the probabilities of the metric-types located in row headers (p_{rh}), located in column headers (p_{ch}), or not located in the headers (p_{capt}), where $p_{rh} + p_{ch} + p_{capt} = 1$.

Metric-type header-level gates Since the attention scores a_{rh_k} and a_{ch_l} capture the relevant header-level information in row and column, these attention scores are used as header-level weights as follows:

$$\mathrm{w}_{hlevel_i} = [a_{rh_k} p_{rh}; a_{ch_l} p_{ch}], \tag{5}$$

where $i \in \{1, ..., u, (u{+}1), ..., (u{+}v)\}$ as a header-level index.

Metric-type generation gates In our pointer-generator network, we use the sigmoid layer to obtain a switch copy probability:

$$p_{copy} = \text{sigmoid}(C_{capt}), \tag{6}$$

which lets us choose between copying word w_{capt} from a table caption and generating word w_m from the metric-type vocabulary, where $p_{copy} \in [0, 1]$. We use a softmax function to compute the probability distribution over the metric-type vocabulary:

$$P_{vocab}(w_m) = \text{softmax}(C_{capt}). \tag{7}$$

Then, we obtain the following probability distribution over the extended vocabulary:

$$P(w_m) = p_{copy} \sum_{i:w_i=(w_m)}^{n} a_{capt_i} + (1 - p_{copy}) P_{vocab}(w_m), \tag{8}$$

where i is the index of metric-type tokens in the vocabulary.

Learning objective For training, we exploit the negative log-likelihood objective as the loss function. In addition, we adopt supervised attention (Liu et al., 2016) for jointly supervising the row and column header-level attention to obtain the metric-type header-level. We combine all loss functions in the location classification and token generation model, and define α as the weight as follows:

$$\mathcal{L} = -((1-\alpha)(\sum_{c} z_{hloc} \log p_{hloc_c} + \sum_{i=1}^{u+v} \log \mathrm{w}_{hlvl_i}) + \alpha(\log p_{copy} + \log P_{vocab}(w_m))), \tag{9}$$

where $c \in \{capt, rh, ch\}$ is the metric-type header-location classes and z_{hloc} is the binary indicator (0 or 1) of each corresponding class.

4.2 Fine-tuning BERT-based Model

Input representation Input text in a fine-tuned BERT-based model is preprocessed by inserting two special tokens, [CLS] and [SEP]. In the original BERT architecture, [CLS] is appended to the beginning of input as the representation of the entire input sequence, and [SEP] is inserted after each input type as a sign of a segment boundary. For example, in question-answering tasks with two

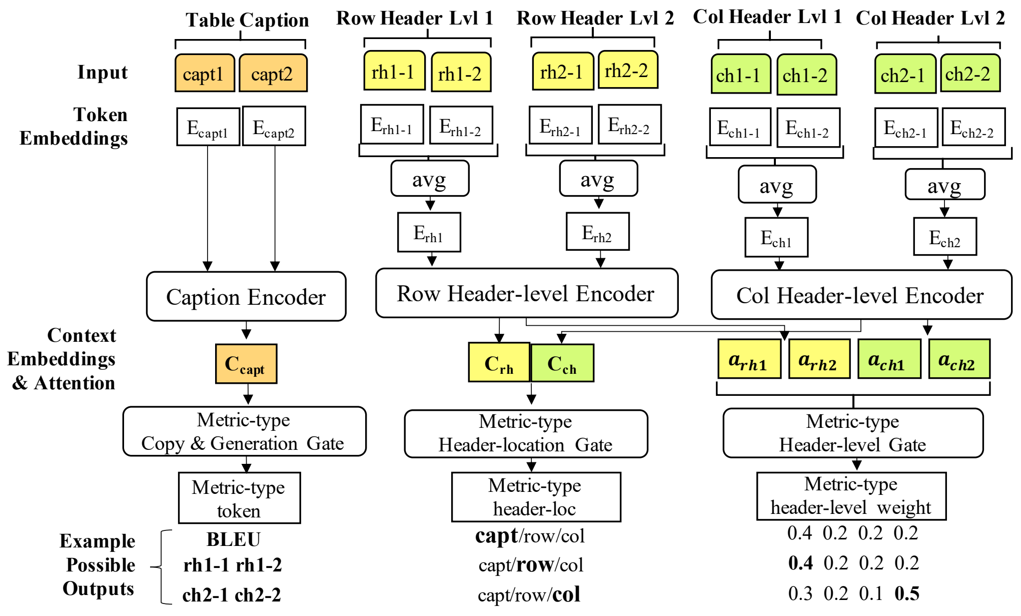


Figure 4: Architecture of proposed pointer-generator-based model to identify metric-types in tables.

types of input text, pairs of question and answer, a [CLS] token is appended before question tokens, and [SEP] tokens are placed after question and after answer tokens, to separate the question and answer segments. Following Liu and Lapata (2019), we customize these preprocessing schemes by inserting [CLS] before each segment and inserting [SEP] after each segment. We divide our inputs into several segments: caption, row header level 1 to u, and column header level 1 to v.

The input text after preprocessing is denoted as a sequence of tokens $X = (x_1, x_2, \cdots, x_n)$. There are three kinds of embedding assigned to each x_i: token embeddings representing the meaning of each token, segmentation embeddings indicating the segment boundaries of a sequence of tokens, and position embeddings covering token position within the sequences. Since BERT only covers two segments in its input, we treat the odd segment as segment A and the even one as segment B. The sum of these three embeddings is fed to a bidirectional Transformer layer of BERT.

We use the token representations from the top hidden layers of the pre-trained Transformer as context embeddings. We assume the context vectors of each [CLS] token can represent the segment sequences better. As shown in Figure 5, we denote the input embedding as E, the final hidden vector of the [CLS] token for the i^{th} input segment as $C_i \in \mathbb{R}_H$, and the final hidden vector for the j^{th} input token as $T_j \in \mathbb{R}_H$.

We use a metric-type header-location gate and a metric-type header-level gate for metric-type location classification, and a metric-type generation gate to generate metric-type tokens from vocabulary covering out-of-header metric-types. Our BERT-based model architecture is shown in Figure 5.

Metric-type header-location gates We feed the first segment context C_1 to the softmax layer to obtain the metric-type header-location probability:

$$p_{hloc} = \text{softmax}(C_1). \tag{10}$$

Metric-type header-level gates In our task, segments are used to represent the table section that is most related to metric-type. We incorporate the segment context C_i to the sigmoid layer to obtain the probability of the metric-type being located in a specific header-level:

$$p_{hlevel_i} = \text{sigmoid}(C_i). \tag{11}$$

The probabilities are then normalized to all segments as a weight score of the header-level:

$$\text{w}_{hlevel_i} = \frac{p_{hlevel_i}}{\sum_{i=1}^{n} p_{hlevel_i}}. \tag{12}$$

Metric-type generation gates We use a softmax function based on the first segment context C_1 to compute a probability distribution over the metric-type vocabulary:

$$P_{vocab}(w_m) = \text{softmax}(C_1). \tag{13}$$

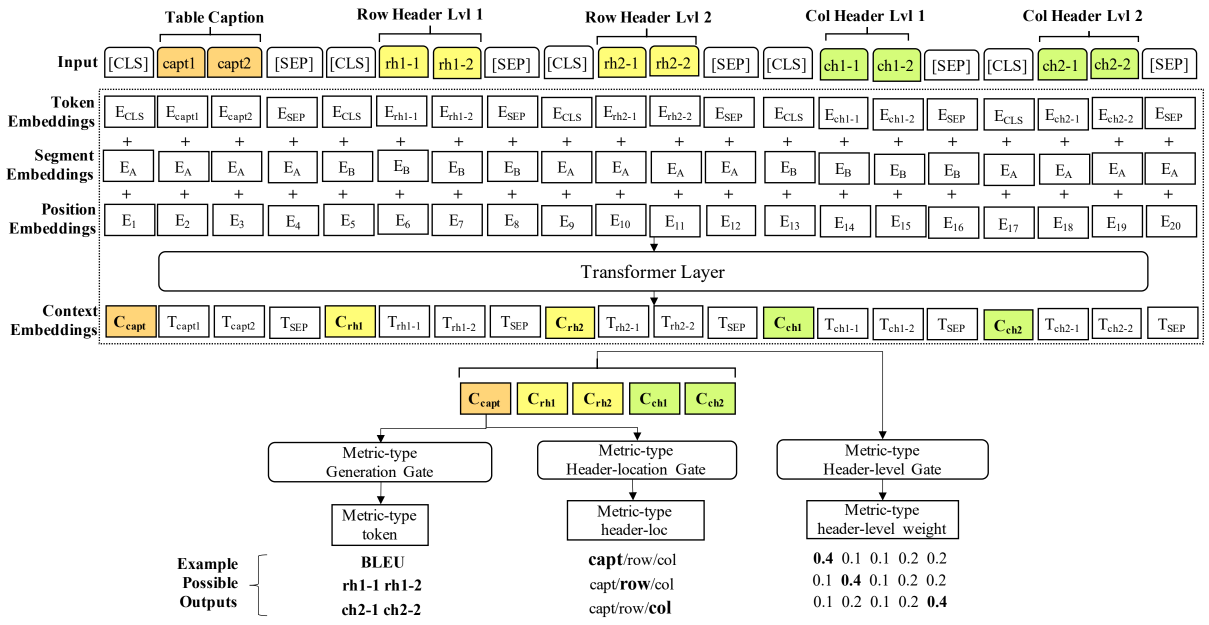


Figure 5: Architecture of proposed BERT-based model to identify metric-types in tables.

Learning objective We combine all loss functions in the metric-type header-location, metric-type header-level, and metric-type generation gates:

$$\mathcal{L} = -((1-\alpha)(\sum_{c} z_{hloc} \log p_{hloc_c} + \sum_{i=1}^{n} \log \mathrm{w}_{hlvl_i}) + \alpha \log P_{vocab}(w_m)), \quad (14)$$

where α is the weight of the metric-type generation functions.

5 Experimental Settings

5.1 Baseline Model

We use two SVM classification models as baselines: a metric-type location prediction model and a metric-type token prediction model from the vocabulary of metric-types. We use tf.idf of the concatenation header name tokens for all levels as input representations in the first model and tf.idf of the caption tokens in the second one. We tuned hyperparameters of the SVM model and reported the best results.

5.2 Metrics Evaluation

We use accuracy metrics to evaluate the metric-type location and generated metric-type tokens.

Metric-type location accuracy The target of the metric-type location prediction model is the metric-type located in the row headers, in the column headers, or not found in the headers. The accuracy of header-location (acc_{hloc}) is the rate of correct header-location predictions.

Since details about the metric-type location in the header-level are needed to identify metric-type token lists, we also compute the accuracy of metric-type header-level (acc_{hlevel}) using the ratio of correct header-level predictions to the total number of predictions.

Metric-type token accuracy Let $\hat{m} = (\hat{w}_{m_1}, ..., \hat{w}_{m_n})$ denote the sequence of predicted metric-type tokens for n_r rows or n_c columns (depending on the header-location prediction), and $m = (w_{m_1}, ..., w_{m_n})$ denote the target ones: for example, $\hat{m} = (\text{f1, f1, f1})$ and $m = (\text{f-1, f-1, f-1})$. We calculate the metric-type token accuracy using string matching of all token lists in $\hat{m}$ and m:

$$acc_{m}^{sm} = \frac{\#\text{ correct } \hat{m}}{\#\ \hat{m}}, \quad (15)$$

and string matching of each token pair $\hat{w}_{m_i}$ and w_{m_i} in the token lists:

$$acc_{m\_token}^{sm} = \frac{\#\text{ correct } \hat{w}_m}{\#\ \hat{w}_m}. \quad (16)$$

To cover token prediction with an abbreviation, we compute the metric-type token accuracy based on the ordered character matching as follows:

$$acc_{m\_token}^{ocm} = \frac{d}{\#\ \hat{w}_m}, \quad (17)$$

Model	acc_{hloc}	acc_{hlevel}	acc^{sm}_{m}	$acc^{sm}_{m\_tok}$	$acc^{ocm}_{m\_tok}$
SVM	81.48	82.96	67.41	69.83	69.83
Pointer-Generator Supervised-Att (Glove)	84.44	84.44	68.89	69.57	72.37
Pointer-Generator Supervised-Att (BERT)	67.41	51.11	45.93	33.66	36.61
Pointer-Generator Supervised-Att (SciBERT)	71.11	57.78	44.44	32.51	35.47
Fine-tuned BERT	91.11	90.37	74.81	77.50	80.46
Fine-tuned SciBERT	**93.33**	**93.33**	**79.26**	**81.61**	**85.06**

Table 2: Test accuracies (%) of different metric-type identification models.

Model	acc_{hloc}	acc_{hlevel}	acc^{sm}_{m}	$acc^{sm}_{m\_tok}$	$acc^{ocm}_{m\_tok}$
Pointer-Generator Supervised-Att (Glove)	84.44	84.44	68.89	69.57	72.37
- copy	85.19	84.44	62.22	62.89	65.52
- copy and generation	82.96	82.22	56.30	54.35	56.98

Table 3: Accuracy scores (%) of ablation test of our pointer-generator-based model.

Model	acc_{hloc}	acc_{hlevel}	acc^{sm}_{m}	$acc^{sm}_{m\_tok}$	$acc^{ocm}_{m\_tok}$
Fine-tuned BERT	91.11	90.37	74.81	77.50	80.46
- segment embeddings	87.41	87.41	72.59	75.70	78.00
Fine-tuned SciBERT	93.33	93.33	79.26	81.61	85.06
- segment embeddings	91.85	91.85	76.30	79.31	81.28

Table 4: Test accuracies (%) of ablated BERT-based model without segment embeddings.

		Predicted			
		LRow	LCol	CCapt	Gen
Actual	LRow	0	0	0	0
	LCol	2	77	1	6
	CCapt	1	6	16	3
	Gen	0	5	0	18

Table 5: Confusion matrix of Pointer-Generator Supervised-Att (Glove) prediction.

		Predicted		
		LRow	LCol	Gen
Actual	LRow	0	0	0
	LCol	0	80	6
	Gen	0	3	46

Table 6: Confusion matrix of Fine-tuned SciBERT prediction.

where d is the number of $\hat{w}_m$ whose characters are all found in w_m in the same order. For example, the predicted token *RG1* is regarded as correct when the reference token is *ROUGE-1*.

5.3 Implementation Details

We implemented our models using the AllenNLP library (Gardner et al., 2018). In our pointer-generator-based model, we used pre-trained word embeddings for initialization and two-layer BiLSTMs with 256 hidden sizes in both the caption and header-level encoders. We used dropout (Srivastava and Hovy, 2014) with the probability $p = 0.1$. For optimization in the training phase, we used Adam as the optimizer with a batch size of 10 and a learning rate of 3×10^{-3} and 3×10^{-5} in pointer-generator-based and BERT-based, respectively, with a slanted triangular schedule (Howard and Ruder, 2018). We trained the model for a maximum of 20 epochs with early stopping on the validation set (patience of 10) and set α to 0.5. We used the original BERT and the domain-specific SciBERT uncased model to fine-tune our BERT-based model.

6 Results

6.1 Experimental Results

Model comparison The performances of the proposed and baseline models are shown in Table 2. We can see that the Pointer-Generator Supervised-Attention model initialized by Glove embeddings outperformed the baseline in predicting metric-type location. The accuracy of this model in the metric-type generation part mostly scored better than the baseline. However, the performances dropped significantly when the input

Caption	experimental results in exploring the shared syntactic order event detector
Header	rh level 1: [model, model, model]
	rh level 2: [cl trans mlp, cl trans cnn, cl trans hbrid]
	ch level 1: [pre., rec., f1]
Gold metric-type	[pre., rec., f1]
Predicted metric-type	[pre., rec., f1]
Caption	results on the image and video datasets of sts task (pearson's r $\times$ 100)
Header	rh level 1: [method, method, method]
	rh level 2: [sts baseline, pivot, ours]
	ch level 1: [ms-vid (2012), pascal (2014), pascal (2015)]
Gold metric-type	[r, r, r]
Predicted metric-type	[pearson's, pearson's, pearson's]

Table 7: Example of table caption, headers, and predicted metric-type.

was initialized by BERT as well as by SciBERT. BERT and SciBERT embeddings failed to guide our pointer-generator-based model in the metric-type identification task, especially in generating metric-type tokens.

The accuracy of our BERT-based model was significantly better than the others, achieving header-location and header-level prediction accuracy of more than 90% and generation accuracy improvement of more than 7 points (%). The fine-tuned BERT-based model using a domain-specific SciBERT led to significant performance gains in all metrics.

The effect of copy mechanism We evaluated our pointer-generator-based model using an ablation test, as shown in Table 3. As we can see, the performances of our generation model without a copy mechanism decreased. This demonstrates that incorporating the copy mechanism is beneficial in a metric-type token generation. Our model had the worst accuracy when it ran without a pointer-generator network since the location prediction model alone failed to handle out-of-header metric-types.

The effect of segment embeddings Table 4 shows the effect of segment embeddings in our BERT-based model. The accuracies of Fine-tuned BERT and the SciBERT model without segment embeddings both decreased. This means that segment embeddings successfully discriminate header-level boundaries in the input representation of BERT-based models.

6.2 Qualitative Analysis

We analyzed the errors of our pointer-generator-based and fine-tuned SciBERT models by means of the confusion matrices shown in Tables 5 and 6. For better understanding, we simply define our outputs in the matrices as "LRow" for metric-type located in row headers, "LCol" for metric-type located in column headers, "CCapt" for metric-type copied from the caption, and "Gen" for metric-type generated from the vocabulary. The matrix for the fine-tuned SciBERT model does not include the CCapt class since this model does not contain a copy mechanism.

As shown in the Table 5, the most correct classifications were for copying from the header (row and column), while the highest confusions were for copying from the caption and generation from the vocabulary. The accuracy of generating correct metric-type tokens from the vocabulary was 27.78%, and the accuracy of copying a metric-type from the caption was 75%. The copying mechanism contributes to a better performance than generation one.

From the confusion matrix of the SciBERT-based model in Table 6, we can see that the highest confusion was for copying from the header. We also computed the accuracy of generated metric-type tokens and found that just 58.7% of the generated tokens were correct.

We also investigated the errors in the predicted metric-type tokens. We found that the models tended to generate more generic metric-types; for example, they extracted *score* as a prediction for the target *accuracy*. On the other hand, our models generated similar terms to the ground truth metric-type, such as generating the metric-type *pearson's* for the target *r*. The examples are shown in Table 7.

7 Conclusion

In this work, we provided multi-level header numerical table datasets extracted from scientific papers consisting of header tables, captions, and metric-types. We introduced a metric-type identification task for multi-level header numerical tables, and proposed joint location prediction and generation models to solve the task. We have shown that our proposed model can identify metric-types from the multi-level header tables, both when the metric-types are included in the headers and when they are not.

Our datasets only cover scientific papers in the computational linguistic domain. The generalization of our results beyond domain still remains an open question due to the difficulties of collecting comparable datasets in other domains without additional annotation by human experts.

Acknowledgements

Lya Hulliyyatus Suadaa is supported by the Indonesian Endowment Fund for Education (LPDP) and Okumura-Takamura-Funakoshi Laboratory, Tokyo Institute of Technology. This work is partially supported by JST PRESTO (Grant Number JPMJPR1655). We thank the anonymous reviewers for helpful discussion of this work and comments on previous drafts of the paper.

References

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Annemarie Friedrich, Heike Adel, Federico Tomazic, Johannes Hingerl, Renou Benteau, Anika Maruscyk, and Lukas Lange. 2020. The SOFC-exp corpus and neural approaches to information extraction in the materials science domain. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1255–1268, Online. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

Yufang Hou, Charles Jochim, Martin Gleize, Francesca Bonin, and Debasis Ganguly. 2019. Identification of tasks, datasets, evaluation metrics, and numeric scores for scientific leaderboards construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5203–5213, Florence, Italy. Association for Computational Linguistics.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

Matthew Prancis Hurst. 2000. *The interpretation of tables in texts*. Ph.D. thesis, The University of Edinburgh.

Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. 2016. Neural machine translation with supervised attention. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3093–3102, Osaka, Japan. The COLING 2016 Organizing Committee.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3730–3740, Hong Kong, China. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Nikola Milosevic, Cassie Gregson, Robert Hernandez, and Goran Nenadic. 2016. Disentangling the structure of tables in scientific literature. In *Natural Language Processing and Information Systems*, pages 162–174, Cham. Springer International Publishing.

Nikola Milosevic, Cassie Gregson, Robert Hernandez, and Goran Nenadic. 2019. A framework for information extraction from tables in biomedical literature. *International Journal on Document Analysis and Recognition (IJDAR)*, 22(1):55–78.

Armineh Nourbakhsh, Mohammad M. Ghassemi, and Steven Pomerville. 2020. Spread: Automated financial metric extraction and spreading tool from earnings reports. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 853–856. ACM.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368.

Shashank Srivastava and Eduard Hovy. 2014. Vector space semantics with frequency-driven motifs. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 634–643, Baltimore, Maryland. Association for Computational Linguistics.