NTCIR-18: Proceedings of the 18th NTCIR Conference on Evaluation ofInformation Access Technologies, June 10-13, 2025, Tokyo, Japan
DOI: https://doi.org/10.20736/0002002096

AKBL at NTCIR-18 U4 TableRetrieval and TableQA

So Takasago
Toyohashi University of Technology
Japan
takasago.so.ro@tut.jp

Tomoyoshi Akiba
Toyohashi University of Technology
Japan
akiba@cs.tut.ac.jp

Abstract

In this paper, we propose a three-stage method for the U4 TableQA task. The method first analyzes and segments the target table into header and data cell sections using a machine learning classifier. Then, it generates natural language descriptions for each data cell using sentence templates based on the table structure. Finally, it retrieves relevant sentences matching the input question from the generated sentence set to form the TableQA result. This approach is also extended to the Table Retrieval task.

Evaluation experiments showed that the Table Retrieval task achieved an accuracy of 0.3569, whereas for the TableQA task, the accuracy of cell_id prediction was 0.7797, and the value prediction was 0.7168.

Keywords

Table QA, table structure recognition, text generation

Team Name

AKBL

Subtasks

Table Retrieval (Japanese), Table Question Answering (Japanese)

1 Introduction

In this paper, we propose a method for the NTCIR-18 U4 [2] TableQA task. The proposed method solves the TableQA task in three stages. In the first stage, we analyze the target table and divide it into header and data cell sections. The header section consists of an inverted L-shaped region comprising the upper and left parts of the table, while the data cell section occupies the remaining lower-right rectangular region (Figure 17). To perform this division, we use a classifier constructed through supervised machine learning that categorizes each cell into four types: Metadata, Header, Attribute, and Data. Based on the classification results for all cells, we determine the boundary between the header and data cell sections.

In the second stage, we generate natural language descriptions for each data cell. For text generation, we use the table divided in the first stage, creating sentences using a template with title text explaining each cell's row and column along with the cell's own value. We perform this for all data cells to create a set of searchable sentences. Finally, we search for sentences matching the question from this set to determine the TableQA result. Additionally, by using this method to generate a representative set of sentences for the entire table, we also address the Table Retrieval task.

The evaluation experiments showed that while the Table Retrieval task achieved a modest accuracy of 0.3287, the TableQA task demonstrated significantly better performance. By employing Run ID 127, which utilizes our best-performing configuration, we achieved a cell_id accuracy of 0.7850 and a value accuracy of 0.6871.

2 Methods

Figure 1 shows the flow of our proposed methods. This section describes our proposed methods for the TableRetrieval and TableQA tasks.

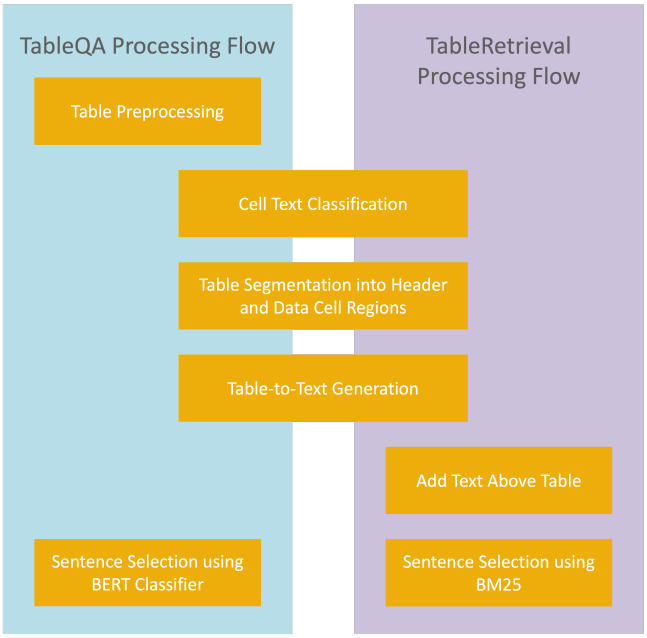

Figure 1: Flow of Proposed Methods for Each Task

2.1 Table Preprocessing

Since the provided annual securities reports are HTML documents, we first extract and preprocess the tables. The extraction and preprocessing are performed in the following steps:

(1) Extract tables from HTML in sequence
(2) Normalize cell text
(3) Process units affecting entire rows/columns
(4) Process units affecting individual cell text
(5) Process units affecting the entire table

First, we extract tables from HTML, considering cell merging during extraction.

Next, we perform cell text normalization. The normalization includes:

- Using provided normalization functions (converting full-width to half-width characters, unifying triangle symbols to minus signs, etc.)
- Standardizing dates to YYYY-MM-DD format
- Converting birth dates (YYYY 年 MM 月 DD 日生) to YYYY-MM-DD format
- Normalizing ” 円” and ” 銭” to decimal values
- Removing annotations like ” ※ 2” and ” 〔9〕 ”

For tables normalized in this way, we process numerical cells according to their specified units.

Initially, we process units affecting entire rows or columns. Figure 10 shows an example where sales and operating profit are specified in ”millions of yen”. To apply such units to numerical cells, we process them as follows: ”If a specific unit exists in a cell, apply that unit to all numerical cells in that row (or column).” The following units were handled:

(百万円) Multiply values by 1,000,000
(%) Multiply values by 0.01
(千円) Multiply values by 1,000
(千株) Multiply values by 1,000

Additionally, there are units that apply only to individual cells. Figure 2 shows an example. When we apply the previously described processing to this table, we get the result shown in Figure 4. At this stage, the cell ” 千人民元 25000” (25,000 thousand Chinese yuan) is not processed as it is not recognized as a numerical cell.

For this table, we apply the following processing: ”If a cell contains ’[specific unit] + number’, remove the unit and apply it to the cell value.” This results in the outcome shown in Figure 6.

名称	住所	資本金(百万円)
(株)ダイエー	長野県	100
AeonMaxvalu	中華人民共和国 広東省	千人民元250000
ミニストップ(株)	千葉県	7491

Figure 2: Example of a Table Before Processing

Name	Address	Capital Stock(Million Yen)
Daiei Co., Ltd.	Nagano Pref.	100
AeonMaxvalu	Guangdong Province, People's Republic of China	Thousand Chinese Yuan (RMB)250000
Ministop Co., Ltd.	Chiba Pref.	7491

Figure 3: Example of a Table Before Processing in English

Here are examples of the units that were targeted:

千シンガポールドル, etc. Multiply values by 1,000
百万ルピア, etc. Multiply values by 1,000,000

In total, we normalized 13 different units in this way.

Finally, we process units affecting the entire table. Figure 8 shows an example where ”(単位: 百万円)” is specified at the top right of the table, affecting all values. This processing was applied only to tables that hadn’t undergone other unit processing, and we only handled ”(単位: 百万円)” in this case.

Name	Address	Capital Stock(Million Yen)
Daiei Co., Ltd.	Nagano Pref.	100000000
AeonMaxvalu	Guangdong Province, People's Republic of China	Thousand Chinese Yuan (RMB)250000
Ministop Co., Ltd.	Chiba Pref.	7491000000

Figure 4: Example After Row Unit Processing in English

Name	Address	Capital Stock(Million Yen)
Daiei Co., Ltd.	Nagano Pref.	100000000
AeonMaxvalu	Guangdong Province, People's Republic of China	Thousand Chinese Yuan (RMB)250000
Ministop Co., Ltd.	Chiba Pref.	7491000000

Figure 5: Example of a Table Before Processing in English

名称	住所	資本金(百万円)
(株)ダイエー	長野県	100000000
AeonMaxvalu	中華人民共和国 広東省	250000000
ミニストップ(株)	千葉県	7491000000

Figure 6: Example After Complete Processing

Name	Address	Capital Stock(Million Yen)
Daiei Co., Ltd.	Nagano Pref.	100000000
AeonMaxvalu	Guangdong Province, People's Republic of China	250000000
Ministop Co., Ltd.	Chiba Pref.	7491000000

Figure 7: Example of a Table Before Processing in English

2.2 Common Method

We performed TableRetrieval and TableQA using the preprocessed tables. To accomplish these tasks, we propose a method that extracts values from tables and tables from documents by ”converting tables into natural language sentences” as a common approach. Specifically, we divided the table into header and data cell sections, and based on this division, we converted the table into natural language text.

First, we classify each cell text in the table into one of four categories: Metadata, Header, Attribute, and Data, following the TDE subtask setting from NTCIR-17 UFO.

The NTCIR-17 UFO TDE subtask was a subtask conducted last year at the NTCIR-17-UFO evaluation forum, which classifies table cell text into four categories: Metadata, Header, Attribute, and Data. This task was designed to understand the structure of tables contained in annual securities reports.

For the TDE implementation, we referenced the approach of OUC [4], who participated in NTCIR-17 UFO. The method uses a

		(単位:百万円)
	2018-02-21	2019-02-21
売上高		
不動産賃貸収入	29545	25963
関係会社受取配当金	57526	4448

Figure 8: Example of Unit Affecting Entire Table

		(Unit: Million Yen)
	2018-02-21	2019-02-21
Net Sales / Revenue		
Real Estate Rental Income	29545	25963
Dividends Received from Affiliated Companies	57526	4448

Figure 9: Example of Unit Affecting Entire Table in English

回次	回次	第44期	第45期
決算年月	決算年月	2017年2月	2018年2月
売上高	(百万円)	458140	512958
経常利益	(百万円)	75007	87563

Figure 10: Example of Unit Affecting Entire Row

Fiscal Term / Period	Fiscal Term / Period	44th Fiscal Term	45th Fiscal Term
Settlement Date	Settlement Date	February 2017	February 2018
Net Sales / Revenue	(Million Yen)	458140	512958
Ordinary Income / Recurring Profit	(Million Yen)	75007	87563

Figure 11: Example of Unit Affecting Entire Row in English

BERT classifier that takes cell text as input and performs four-way classification.

Based on the TDE results, we perform table segmentation. Rather than using a single label, the TDE output uses a 4-label probability vector (Pm_i, Ph_i, Pa_i, Pd_i) to determine the segmentation.

To divide the table into header (upper-left) and data cell (lower-right) regions, we consider 100 different segmentation patterns from row 1, column 1 to row 10, column 10. For each segmentation pattern, we calculate the TSS (Score) as shown in the following equation:

$$TSS = \frac{\sum_{i \in \mathcal{H}} (Pm_i + Ph_i + Pa_i)}{|\mathcal{H}|} + \frac{\sum_{j \in \mathcal{D}} Pd_j}{|\mathcal{D}|} \quad (1)$$

Here, $\mathcal{H}$ represents the set of cells divided into the header section, and $\mathcal{D}$ represents the set of cells divided into the data cell section. $Pm_i + Ph_i + Pa_i$ is the sum of probabilities for Metadata, Header, and Attribute for the i-th cell assigned to the header section. Similarly, Pd_j represents the Data probability for the j-th cell assigned to the data cell section. We calculate the TSS by taking the sum of the average probability that header section cells are classified as Metadata, Header, or Attribute and the average probability that data cell section cells are classified as Data, and adopt the segmentation pattern that yields the highest TSS value. Note that when the source text for classification is an empty string, we consider the classification result unreliable and do not use it in TSS calculation.

When we classify Figure 12 using TDE, we obtain the result shown in Figure 14. In the example, probabilities are truncated to 4 decimal places.

回次	第44期
売上高	458140
経常利益	75007

Figure 12: Example of Table Before TDE Processing

Fiscal Term / Period	44th Fiscal Term
Net Sales / Revenue	458140
Ordinary Income / Recurring Profit	75007

Figure 13: Example of Table Before TDE Processing in English

[0.001, 0.996, 0.002, 0.000]	[0.000, 0.013, 0.983, 0.023]
[0.001, 0.994, 0.005, 0.000]	[0.000, 0.000, 0.000, 0.999]
[0.000, 0.987, 0.009, 0.001]	[0.000, 0.000, 0.000, 0.999]

Figure 14: Example of Table After TDE Processing

The cells in Figure 14 contain probability vectors [Pm, Ph, Pa, Pd]. For this figure, we consider two possible segmentation patterns: 1 row × 1 column and 2 rows × 1 column, from which we determine the optimal segmentation. Figures 15 and 16 show the tables segmented at 1×1 and 2×1, respectively.

The TSS for the 1×1 segmentation is:

$$\frac{(0.001 + 0.996 + 0.001) + \ldots + (0.000 + 0.987 + 0.009)}{4} + \frac{0.999 + 0.999}{2} = 1.99675 \quad (2)$$

And for the 2×1 segmentation:

$$\frac{(0.001 + 0.996 + 0.001) + \ldots + (0.000 + 0.987 + 0.009)}{4} + 0.999 = 1.7972 \quad (3)$$

Therefore, we determined that the 1×1 segmentation was optimal for this table.

[0.001, 0.996, 0.002, 0.000]	[0.000, 0.013, 0.983, 0.023]
[0.001, 0.994, 0.005, 0.000]	[0.000, 0.000, 0.000, 0.999]
[0.000, 0.987, 0.009, 0.001]	[0.000, 0.000, 0.000, 0.999]

Figure 15: Table Segmented at 1×1

[0.001, 0.996, 0.002, 0.000]	[0.000, 0.013, 0.983, 0.023]
[0.001, 0.994, 0.005, 0.000]	[0.000, 0.000, 0.000, 0.999]
[0.000, 0.987, 0.009, 0.001]	[0.000, 0.000, 0.000, 0.999]

Figure 16: Table Segmented at 2×1

2.3 Table-to-Text Generation

Next, we perform text generation using the segmented tables. For each target cell, we generate text using a template ”X の Y は Z です。” where X is the concatenation of header texts from the top with ”、”, Y is the concatenation of header texts from the left with ”、”, and Z is the target cell value. Note that when the target cell Z is empty, we skip generating text for that cell.

For example, when generating text for the data cell ”458140” in Figure 17, X is ” 第 44 期”, Y is ” 売上高、（百万円）”, and Z is ”458140”, resulting in the generated text ” 第 44 期の売上高、(百万円) は 458140 です。”

Additionally, considering tables where column and row relationships may be reversed, we also explored a method that generates two sentences using both templates ”X の Y は Z です。” and ”Y の X は Z です。”

回次	回次	第44期
決算年月	決算年月	2017年2月
売上高	(百万円)	458140
経常利益	(百万円)	75007

Figure 17: Table After Segmentation

2.3.1 Proposed Method for TableRetrieval. In TableRetrieval, given a question text and document ID, the task is to output a table ID. To accomplish this, we extract all tables from the HTML file of the given document ID, generate sentences for each table, and combine them into a single document to create document representations of tables.

Fiscal Term / Period	Fiscal Term / Period	44th Fiscal Term
Settlement Date	Settlement Date	February 2017
Net Sales / Revenue	(Million Yen)	458140
Ordinary Income / Recurring Profit	(Million Yen)	75007

Figure 18: Table After Segmentation in English

Using these generated documents and question text, we implemented table retrieval using Okapi BM25 for search.

While training, validation, and test datasets were provided, we only used the test data for implementation and evaluation without performing any training.

We performed processing and validation through the following steps:

(1) Extract tables from HTML
(2) Generate sentences using first row and first column as header cells
(3) Search through table documents using question text with Okapi BM25
(4) Output table_id based on search results

The BM25 parameters were set to $k_1 = 1.5$ and $b = 0.75$.

2.3.2 Proposed Method for TableQA. In TableQA, given a question text and table ID, the task is to output a cell ID and its value. To accomplish this, we extract the corresponding table from the HTML based on the given table ID, perform table segmentation, and generate text for each cell through text generation.

Using these generated sentences and the question text, we select the appropriate cell and value.

For sentence selection, we use a binary classifier that determines whether a query and sentence are relevant. Specifically, we construct a BERT classifier that takes the concatenation of the query and the sentence text with a [SEP] token as input and outputs a binary classification result of relevant (1) or not relevant (0). The [SEP] token is one of the special tokens used in BERT models, indicating a ”Separator” between sentences. Here is an example of the input text:

Example of Concatenated Text

株式会社ニトリホールディングスの第 44 期における「売上高、経営指標等」は？ [SEP] 第 44 期の売上高、(百万円) は 458140 です。

For training data, we randomly selected one matching sentence (relevant) and one non-matching sentence (irrelevant) for each query text from the correct cell and incorrect cells respectively. We created one piece of training data each for relevant and irrelevant labels per query text, creating training data for all query texts provided in TableQA. Table 1 shows the number of question texts in the training data provided by NTCIR-18 U4’s TableQA subtask and the number of fine-tuning training data created from it. The difference in training data between DryRun and FormalRun is due

to the removal of data deemed inappropriate during the DryRun period.

In theory, the number of relevant and irrelevant sentences should equal the number of given question texts. However, this discrepancy in the results occurred due to cases where either there were no data sections in the table segmentation (tables where all cell text was classified as header text by TDE) or where answer cells were included in the header section.

We used ku-nlp/deberta-v3-base-japanese, a pre-trained BERT model provided by Kyoto University Language Media Processing Laboratory.

Table 1: Number of Questions and Fine-tuning Training Data in TableQA Subtask

Dataset Type	DryRun	FormalRun
Number of Questions in Training Data	22982	10300
Generated Relevant Sentences	22152	10090
Generated Irrelevant Sentences	22969	10299

3 Experiments

In this section, we present the evaluation results.

3.1 TDE Evaluation

For training data, we used the FormalRun data from NTCIR-17 UFO. We implemented the model by fine-tuning the BERT pre-trained model cl-tohoku/bert-large-japanese-v2 with cell text and cell labels obtained from the training data. Table 2 shows the resulting training data.

Table 2: Number of Training Data for TDE

Label	Data Count
Metadata	143
Header	13930
Attribute	11236
Data	41060

Using this training data, Table 3 shows the results of our implemented model. For inference, we used the Test data included in the FormalRun.

Table 3: Experimental Results for TDE

Label	Precision	Recall	F1-score
header	0.8518	0.8623	0.8570
attribute	0.7892	0.7908	0.7900
data	0.9761	0.9718	0.9739
metadata	0.5909	0.5909	0.5909

The target data consists of 192 tables that are subjects of the NTCIR18-U4 TableQA subtask, for which we performed annotation. Figure 19 shows the table segmentation positions for these 192 annotated tables.

The results show that:

- The most common segmentation position is at row 2, column 1
- Most column segmentation positions are at column 1
- The maximum column segmentation position is at row 10

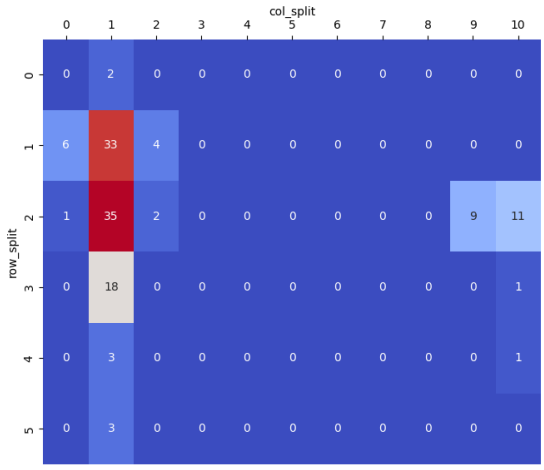

Figure 19: Heatmap of Table Segmentation Positions

Using the created table segmentation dataset, we evaluated the table segmentation performance. Table 4 shows the evaluation results of the table segmentation.

The table segmentation range refers to the search range used when segmenting tables based on TDE results using TSS. For range 1-3, we search through 9 patterns from row 1, column 1 to row 3, column 3, and for range 1-10, we search through 100 patterns from row 1, column 1 to row 10, column 10.

Additionally, the accuracy is shown for cases where empty cell classification results are either used or ignored in TSS calculation.

As shown in Figure 19, since some tables have headers up to column 10, expanding the search range to 1-10 increases accuracy by 0.046. Furthermore, by excluding empty cell classification results from TSS calculation, accuracy increases by 0.109. This is because the TDE model tends to classify empty cells (blank cells) as Data cells, which can decrease accuracy when empty cells appear in the header section.

Table 4: Table Segmentation Accuracy Results

Empty Cell Classification	Segmentation Search Range	Accuracy
Used	1-3	0.605
	1-10	0.651
Ignored	1-10	0.760

3.2 Evaluation of TableRetrieval

Table 5 shows the evaluation results for TableRetrieval. We used validation data for evaluation, with 2,897 question texts as targets.

Table 5: TableRetrieval Evaluation Results (Validation Data)

Method	Segmentation Range	Accuracy
Baseline	Fixed (1×1)	0.181
w/o Normalization	Fixed	0.195
w/o Normalization	1-3	0.209
w/ Table Header Text	1-3	0.314

The baseline shows the results of applying normalization functions and row/column unit processing provided by U4 to cell texts during table preprocessing.

Without performing normalization and by calculating TSS using TDE to search for table segmentation, the accuracy improved from 0.181 to 0.209, an increase of 0.028. Additionally, by adding a sentence above the table based on the HTML, the accuracy reached 0.314.

We then conducted evaluation on the test data. The TaskOrganizer achieved an accuracy of 0.234 by embedding cell text concatenations using text-embedding-3-small and returning answers based on the shortest distance to question embeddings.

Using our best-performing method from the validation data, we achieved an accuracy of **0.3569** on the test data. This result is 0.1435 higher than the TO's result of 0.2134.

3.3 Evaluation of TableQA

Table 6 shows the evaluation results for TableQA. Test data was used for evaluation.

The TaskOrganizer provided the cell text extracts combined with the question text to GPT-4o, which produced answers with a value accuracy of **0.690**.

The Baseline shows the results of applying the normalization functions and row/column unit processing provided by U4 to cell text during table preprocessing. The BERT classifier (JRTEC) refers to a classifier fine-tuned using JRTE-Corpus (Japanese Realistic Textual Entailment Corpus) [1]. JRTE-Corpus is a textual entailment recognition dataset created from Jaran review data and provided by Recruit Co., Ltd.

By expanding the table segmentation search range from 3 to 10, we observed improved accuracy. Additionally, using the textual entailment recognition model yielded slightly higher accuracy compared to BM25.

When fine-tuning the BERT classifier (DeBERTa-V3) using our created training data, we observed significant accuracy improvements compared to using BM25. This result was 0.0401 higher compared to the TO's result of 0.6470. However, when using two sentences during inference, there was a slight decrease in accuracy.

For comparison, Table 7 shows the results of training with the FormalRun dataset. We compared using the method from ID 128.

Comparing the results, we found that the DryRun accuracy was better. Although the FormalRun training data was of higher quality compared to DryRun, we believe these results occurred due to the large difference in the amount of training data between DryRun and FormalRun.

Table 6: Evaluation Results for TableQA (Using DryRun Training Data)

Sentence Selection Method	Method	Table Segmentation Search Range	ID	Accuracy	
				cell_id	value
BM25	Baseline	Fixed (1 row 1 column)	22	0.1326	0.0425
	Improved Normalization and Unit Processing	1-3	31	0.2193	0.2224
		1-10	46	0.2529	0.2735
	Exclude Empty Cells from TSS Calculation	1-10	47	0.2501	0.2727
	Exclude Empty Cells from TSS and Text Generation	1-10	48	0.2925	0.3128
BERT Classifier (JRTEC)	Exclude Empty Cells from TSS and Text Generation	1-10	64	0.3323	0.3179
BERT Classifier (U4 Dataset)	Exclude Empty Cells from TSS and Text Generation	1-10	127	0.7850	0.6871
BERT Classifier (U4 Dataset)	Two-sentence Generation at Inference Only	1-10	128	0.7824	0.6834

Table 7: Accuracy Changes in U4 TQA (Comparing Training Datasets)

Training Dataset	Accuracy	
	cell_id	value
DryRun	0.7824	0.6834
FormalRun	0.7016	0.6301

4 Discussion

In this section, we discuss our findings based on the experimental results from each dataset.

4.1 Table Segmentation and Text Generation

In this research, we performed table segmentation based on TDE output. However, there are various types of tables in the target data, not just simple tables [3], such as:

- Tables with headers that modify header sections
- Tables where multiple headers are combined into a single cell
- Tables where cells contain sentences with multiple pieces of information

With our current method, tables with special structures due to merged cells cannot be properly converted to text. We believe that improving this aspect would enable more accurate text generation.

4.2 TableRetrieval Subtask

In the TableRetrieval subtask, we implemented table search by "combining sentences for each table and comparing them with question text using BM25."

We believe that replacing BM25 with dense vector retrieval methods such as Transformer-based approaches could potentially yield higher accuracy.

4.3 TableQA Subtask

Based on the observations in Section 4.1, we believe that improving the text generation component could lead to more accurate text generation and selection.

Currently, we use a template "X の Y は Z です。" for text generation. However, this can result in unnatural sentences for certain types of tables (such as those where data cells contain sentences).

We believe that incorporating Text-to-Text methods during text generation could help reduce the generation of unnatural sentences.

5 Conclusions

Using our proposed method, we achieved an accuracy of 0.3287 in the TableRetrieval subtask, while in the TableQA subtask, we achieved a cell coordinate accuracy of 0.7580 and a value accuracy of 0.6871. Although the TableRetrieval subtask showed modest results, in the TableQA subtask, we exceeded the TaskOrganizer's GPT-4o results by 0.021.

For future work, we expect to focus on:

- Text generation for tables with complex structures
- Improving the TableRetrieval subtask by incorporating methods from the TableQA subtask
- Reducing unnatural sentence generation by incorporating Text-to-Text methods in the text generation process

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 23K11118.

References

[1] Yuta Hayashibe. 2020. Japanese Realistic Textual Entailment Corpus. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association, Marseille, France, 6827–6834. https://aclanthology.org/2020.lrec-1.843/

[2] Yasutomo Kimura, Eisaku Sato, Kazuma Kadowaki, and Hokuto Ototake. 2025. Overview of the NTCIR-18 U4 Task. *Proceedings of the 18th NTCIR Conference on Evaluation of Information Access Technologies* (6 2025).

[3] Kazuki Okuyama and Yasutomo Kimura. 2024. Analysis of Machine-Unreadable Table Structures in Financial Reports in Japanese. In *Proceedings of the 30th Annual Conference of the Japanese Association for Natural Language Processing (NLP2024)*. P3–20. Summary national conference.

[4] Eisaku Sato, Keiyu Nagafuchi, Yuma Kasahara, Kazuma Kadowaki, and Yasutomo Kimura. 2023. OUC at NTCIR-17 UFO TDE and TTRE. NII Institutional Repository. The OUC team participated in the Table Data Extraction (TDE) subtask and the Text-to-Table Relationship Extraction (TTRE) of NTCIR-17 Understanding of Non-Financial Objects in Financial Reports (UFO). In this paper, we report our methodology in this task and discuss the official results..