Tree-of-Report: Table-to-Text Generation for Sports Game Reports with Tree-Structured Prompting

Anonymous ACL submission

Abstract

Generating sports game reports from structured table data is a challenging table-to-text generation task that requires balancing structured data comprehension with narrative storytelling. While model-based approaches demand large training datasets, prompt-based methods with large language models (LLMs) often suffer from hallucination issues due to poor table comprehension. To address these challenges, we propose **Tree-of-Report**, a novel framework inspired by the "divide and conquer" concept of merge sort, which divides the task into three stages: Content Planning, Operation Execution, and Content Generating. Our method decomposes large tables into smaller sub-tables using a hierarchical tree structure, enabling more effective table comprehension. Additionally, it merges and rewrites texts to produce more detailed and coherent long-form outputs. Experimental results on the RotoWire, MLB, and ShuttleSet+ datasets show that Tree-of-Report outperforms existing prompt-based baselines with relatively lower time and cost, demonstrating its advantage in both effectiveness and efficiency. In summary, this work sets a new precedent for prompt-based table-to-text generation in sports game reports.

1 Introduction

Writing sports game reports requires journalists to analyze match data and craft engaging reports under tight deadlines. Beyond conveying scores and player performance, they must construct compelling narratives that highlight key moments. Automating this process could greatly improve the efficiency and accessibility of sports journalism. However, converting structured match data into natural language remains challenging. Sports reporting also demands adherence to journalistic conventions, integrating game flow, player dynamics, and contextual insights, which require reasoning and advanced text organization skills.

Thus, sports game report generation is a complex table-to-text generation task involving not only data transformation but also discourse structuring, content selection, and information organization. Effectively generating sports articles requires balancing structured data processing with the storytelling aspects of journalism to ensure accuracy and readability. In this study, we focus specifically on the sports domain, utilizing datasets such as RotoWire (Wiseman et al., 2017), MLB (Puduppully et al., 2019b), and ShuttleSet+. These datasets are characterized by high data fidelity and longer textual outputs, making the task more challenging. Figure 1 presents an example from ShuttleSet+. The text contained in the tables is highlighted in **bold**, with different colors used to distinguish information from different tables. This figure also showcases the high data fidelity and long-form text characteristic of sports game reports.

For this task, numerous model-based methods have been proposed, such as NCP (Puduppully et al., 2019a), NDP (Chen et al., 2021), DUV (Gong et al., 2020), Macro (Puduppully and Lapata, 2021), and SeqPlan (Puduppully et al., 2022). However, these approaches require large amounts of training data, making them impractical when datasets are scarce and costly to collect. With advancements in large language models (LLMs), prompt-based methods have become increasingly popular, including Zero-shot, One-shot, and Few-shot learning (Brown et al., 2020), as well as techniques like Chain-of-Thought (Wei et al., 2022), Tree-of-Thought (Yao et al., 2023), and Chain-of-Table (Wang et al., 2024). Although these methods are widely used, in the task of generating sports game reports, LLMs fail to effectively analyze and comprehend information within tables, resulting in the generation of text that is inaccurate or not present in the tables, commonly known as the hallucination issue.

To address these challenges, we propose a novel

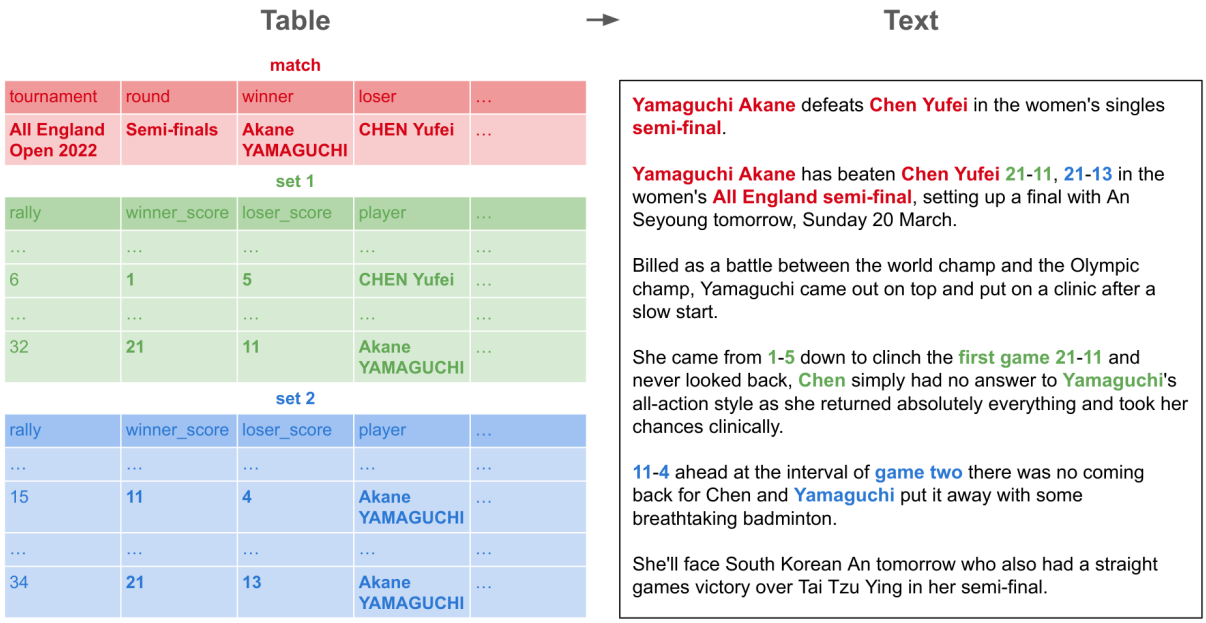

Figure 1: An example from ShuttleSet+, which includes multiple structured tables containing match data along with a corresponding human-written report for the game.

approach, Tree-of-Report, which divides the task into three stages: Content Planning, Operation Execution, and Content Generating. First, in the Content Planning stage, the LLM plans the operations for child nodes based on the table structure. Second, in the Operation Execution stage, these selected operations are executed respectively to update the table, which is then passed to the child nodes. Finally, in the Content Generating stage, the LLM generates text based on the table and returns it to the parent node, which then utilizes the LLM to merge and rewrite these texts into a new text.

Tree-of-Report effectively leverages a hierarchical tree structure to decompose large tables into smaller sub-tables, enhancing the LLM's ability to comprehend tabular information. Additionally, we employ a merge-and-rewrite approach to generate longer and more comprehensive reports. Experimental results demonstrate that Tree-of-Report outperforms other prompt-based baselines on the RotoWire, MLB, and ShuttleSet+ datasets. Furthermore, with optimizations, our method achieves lower time and cost compared to Tree-of-Thought and Chain-of-Table, highlighting its advantages in both effectiveness and efficiency.

We summarize the three main contributions of this paper:

- In the task of table-to-text generation for sports game reports, we introduce Tree-of-Report, a novel framework that recursively decomposes tables into smaller sub-tables, generates short textual descriptions for each sub-table, and merges these short texts into a complete report.
- We introduce a new sports report dataset, ShuttleSet+, containing rally-level data from 58 badminton matches along with the corresponding human-written reports.
- Tree-of-Report outperforms other prompt-based baselines on the RotoWire, MLB, and ShuttleSet+ datasets while maintaining relatively lower time and cost, demonstrating its superiority in both effectiveness and efficiency.

2 Related Work

2.1 Table-to-Text Generation

The goal of the table-to-text generation task is to convert structured tables into unstructured text, typically following a two-stage process: Content Planning and Content Generating (Lin et al., 2024). Content Planning, or "What to say," involves analyzing and filtering the given structured data, selecting relevant information for abstraction and association. Content Generating, or "How to say," focuses on accurately and fluently describing the selected data using natural language. Tree-of-Report follows this principle in its architectural design.

There are numerous datasets available for table-to-text generation, such as WikiBio (Lebret et al.,

2016), ToTTo (Parikh et al., 2020), and TabFact (Chen et al., 2020). Nevertheless, this paper focuses on domain-specific datasets, particularly for sports game reports. For instance, RotoWire (Wiseman et al., 2017) is a dataset consisting of human-written summaries of NBA basketball games paired with their corresponding box and line scores. MLB (Puduppully et al., 2019b) provides baseball statistics accompanied by human-authored summaries from the ESPN website. ShuttleSet+, derived from ShuttleSet22 (Wang et al., 2023), contains rally-level data from 58 badminton matches along with corresponding human-written reports. These datasets share two common characteristics: high data fidelity and long textual outputs. High data fidelity ensures accurate and important information, enabling readers to gain a deeper understanding of the sports matches. Long textual outputs, on the other hand, provide rich and vivid descriptions, enhancing reader engagement and interest in the sports games.

2.2 Model-based Methods

Several previous studies have introduced model-based approaches for table-to-text generation. NCP (Puduppully et al., 2019a) employs a two-stage framework, first generating a content plan that specifies what information to include and in what order before passing it to the text generation stage. NDP (Chen et al., 2021) dynamically selects relevant information from the input data during text generation. DUV (Gong et al., 2020) enhances neural content planning by incorporating contextual numerical value representations for improved value comparison, and applying policy gradient to verify the importance and order of selected records. Macro (Puduppully and Lapata, 2021) introduces a macro planning stage prior to text generation, where macro plans structure key entities, events, and their interactions. SeqPlan (Puduppully et al., 2022) employs a structured variational model to infer latent plans sequentially, interleaving planning and generation steps.

However, under our experimental setup, the dataset size is limited (e.g., ShuttleSet+ contains only 58 instances), making it infeasible to train a model. Therefore, we explore prompt-based methods as an alternative approach.

2.3 Prompt-based Methods

With the rise of LLMs, prompt-based methods have gained increasing attention. Brown et al. (2020) first introduced the Zero-shot, One-shot, and Few-shot approaches, demonstrating that providing some reference examples enables LLMs to achieve strong performance across various tasks. Chain-of-Thought (Wei et al., 2022) enhances LLM reasoning by incorporating a series of intermediate reasoning steps within the prompt. Tree-of-Thought (Yao et al., 2023) enables LLMs to make deliberate decisions by exploring multiple reasoning paths, self-evaluating choices, and backtracking when necessary to optimize global decision-making. Chain-of-Table (Wang et al., 2024) guides LLMs to iteratively generate operations, updating the table to form a tabular reasoning chain, allowing for dynamic operation planning based on previous results.

However, previous methods directly input the entire table into the LLM, making it difficult for the model to fully understand the table structure, thereby resulting in hallucination and failing to ensure high data fidelity. Similarly, these methods directly output the final text in a single step, limiting the model's ability to process comprehensive information, thus failing to produce sufficiently long and detailed textual outputs. Therefore, we propose Tree-of-Text to address these challenges.

3 Tree-of-Report

3.1 Overview

In the task of table-to-text generation, the input consists of multiple tables T, and the output is a textual description t of these tables. We propose a method called Tree-of-Report, inspired by the "divide and conquer" concept of merge sort (Bron, 1972), which constructs a tree structure and divides the task into three stages: Content Planning, Operation Execution, and Content Generating.

In the Content Planning stage, the LLM determines the operations and arguments OA for the child nodes based on the input tables T, the operation history OH, the operation pool OP, and the level L. The number of child nodes must not exceed the maximum degree MAX_DEGREE.

In the Operation Execution stage, the operations are executed sequentially to update T, OH, OP, and L, which are then passed to the child nodes, respectively. This process continues recursively until either a `write()` operation is encountered, or the level L reaches the maximum depth MAX_DEPTH.

In the Content Generating stage, a short text t'

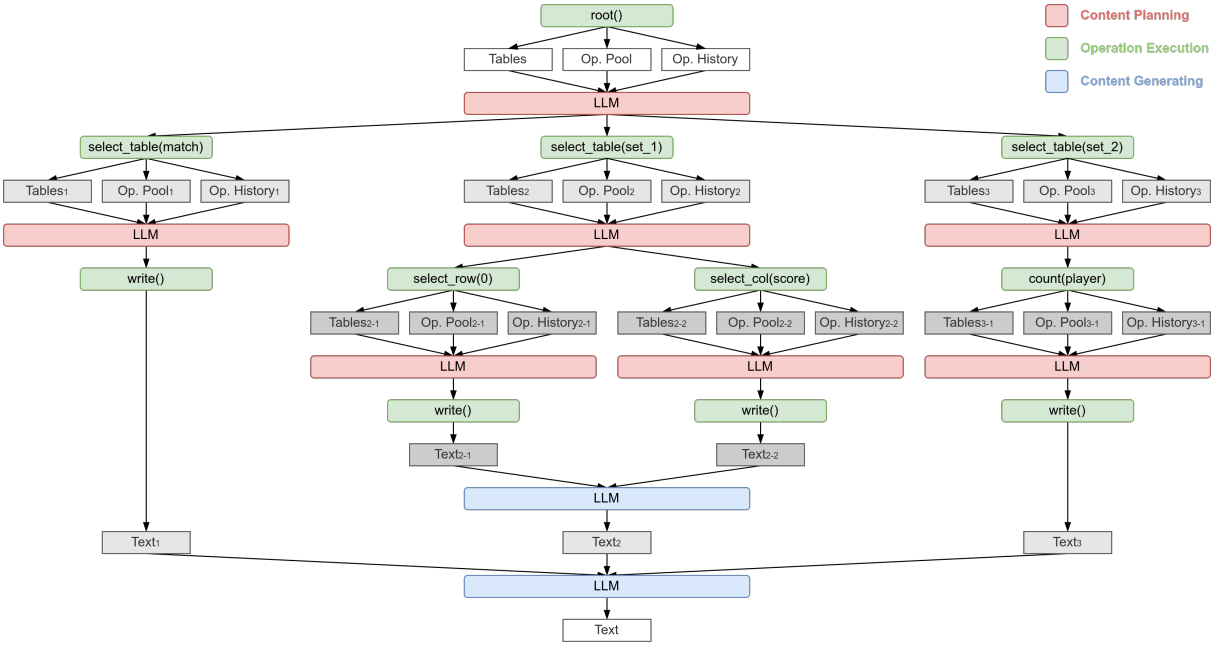

Figure 2: The overall workflow diagram of Tree-of-Report, which divides the Table-to-Text Generation task into three stages: Content Planning, Operation Execution, and Content Generating. To simplify the illustration, we use a simple tree structure as an example; in practice, the tree structure is more complex.

is first generated based on the updated tables T' and returned to the parent node. The LLM then merges and rewrites these texts into a new text t, continuing this process until returning back to the root node.

The overall workflow of Tree-of-Report is illustrated in Figure 2, and the algorithm is presented in Algorithm 1.

3.2 Content Planning

Starting from the root node, the inputs consist of the initial tables $T \leftarrow (T^j \mid j = 1, 2, \ldots, n)$, the operation history $OH \leftarrow (op \mid op = \text{root}())$, the operation pool $OP \leftarrow (op \mid op \in \text{operations},\ op \neq \text{root}())$, and the level $L \leftarrow 0$. Based on these inputs, the LLM determines the operations and arguments for the child nodes, denoted as $OA \leftarrow (O_i(A_i) \mid O_i \in \text{OP}, i = 1, 2, \ldots, d)$, where d represents the degree of this node and must not exceed the maximum degree MAX_DEGREE. The prompt for Content Planning is provided in Appendix A.1.

3.3 Operation Execution

To execute operations that split large tables into smaller ones or generate textual descriptions, we define a total of eight operations as follows:

- `root()`: Does nothing; represents the root node of the tree.
- `select_table()`: Selects a table by its table name.
- `select_row()`: Selects rows based on their row indices.
- `select_col()`: Selects columns based on their column names.
- `count()`: Counts the number of unique values in the specified columns of the tables.
- `sort()`: Sorts rows based on the specified column names and sorting orders.
- `filter()`: Filters rows based on column names, comparison symbols, and values.
- `write()`: Generates text based on the tables using the LLM; represents the leaf node of the tree. The prompt for the `write()` operation is provided in Appendix A.2.

The operations in OA are then executed respectively to update T, OH, OP, and L, where $T_i \leftarrow O_i(T, A_i)$, $OH_i \leftarrow OH + O_i(A_i)$, $OP_i \leftarrow OP - O_i()$, $L_i \leftarrow L + 1$. The updated T_i, OH_i, OP_i, and L_i are then passed to the child nodes, and the process continues recursively until either a `write()` operation is encountered or the level L reaches the maximum depth MAX_DEPTH.

When the level L reaches the maximum depth MAX_DEPTH, we directly call `write()`, where the LLM generates a short text t describing the current input table T and returns t to the parent node. Similarly, when encountering a `write()` operation, the LLM is invoked to generate a short text

t'_i describing the current input table T. Since child nodes also return texts t'_i, we collect them into a sequence $t' = (t'_i \mid i = 1, 2, \ldots, d)$.

3.4 Content Generating

The LLM then merges and rewrites t' into a new text t, which is passed to the parent node. This recursive process continues until it returns to the root node. The text t returned from the root node is the final output. The prompt for Content Generating is provided in Appendix A.3.

For efficiency considerations, we implemented additional optimizations. First, unlike Chain-of-Table, which generates operations first and then arguments, our method generates operations and arguments in one step. Second, if a node has a degree of one, there is no need to use the LLM for merging; the single text can be directly returned. Finally, we experimented with an approach where the LLM is used for merging only at the root node, while other nodes simply concatenate texts. With these optimizations, Tree-of-Report significantly reduces both time and cost.

4 Experiment

4.1 Dataset

4.1.1 RotoWire

The RotoWire (Wiseman et al., 2017) dataset comprises human-written NBA basketball game summaries in English paired with their corresponding box and line scores. These summaries, sourced from *rotowire.com*, are relatively general compared to other datasets, providing more high-level information. The dataset includes 4,853 unique summaries covering NBA games played between January 1, 2014, and March 29, 2017, with some games featuring multiple summaries. The dataset is randomly divided into training, validation, and test sets, containing 3,398, 727, and 728 summaries, respectively. Data preprocessing for RotoWire is provided in Appendix B.1 for further details.

4.1.2 MLB

The MLB (Puduppully et al., 2019b) dataset contains baseball statistics paired with human-written summaries in English sourced from the ESPN website. Compared to RotoWire, it is approximately five times larger, featuring a broader vocabulary and longer summaries. The dataset is divided into 22,821 training, 1,739 validation, and 1,744 testing instances. Data preprocessing for MLB is delivered in Appendix B.2 for further details.

4.1.3 ShuttleSet+

We introduce a new dataset, ShuttleSet+, derived from ShuttleSet22 (Wang et al., 2023). ShuttleSet22 is a human-annotated, stroke-level singles dataset for badminton tactical analysis, comprising 140 sets, 3,992 rallies, and 33,612 strokes from 58 matches played between 2018 and 2022. The dataset features 35 top-ranking men's and women's singles players. Since ShuttleSet22 does not include corresponding textual reports for each match, we collected human-written reports in English for each game from online sources such as the BWF and Olympics websites, and renamed the dataset as ShuttleSet+. Compared to RotoWire and MLB, ShuttleSet+ has fewer data samples, representing a low-resource scenario. In addition, ShuttleSet+ reports contain more detailed information, including rally-level data and tactical analysis. Finally, we randomly split the dataset into training, validation, and test sets using a 40:9:9 ratio. Data preprocessing for ShuttleSet+ is given in Appendix B.3 for further details.

4.2 Evaluation Metric

To quantify the similarity of information between two texts, we use the Information Extraction (IE) metrics introduced by Wiseman et al. (2017). These metrics are based on the output of an IE model, which extracts relation pairs, formatted as `(table|column|value)`, from the generated summary.

Due to the lack of sufficient data to train a new IE model for ShuttleSet+, we propose an alternative approach that leverages the LLM as a substitute for the IE model. To validate its reliability, we manually annotated a set of relations and compared them with those extracted by the LLM, finding that it achieved over 60% on all evaluation metrics. Based on this experiment, we consider that using an LLM as an IE model is a reliable alternative. The full experimental results are provided in Appendix C.

In the experiment, let $\hat{t}$ denote the model output and t symbolize the gold text. Relation Generation (RG) evaluates the count (#) and precision (P%) of relations extracted from $\hat{t}$ that are present in the input table T, representing the amount and accuracy of information in the generated text. Content Selection (CS) assesses the precision (P%), recall (R%), and F1 score (F%) of relations extracted

RotoWire	RG #	RG P%	CS P%	CS R%	CS F%	CO DLD%	Avg.	Time	Cost
Zero-shot	29.69	**97.54**	56.70	55.77	50.85	30.77	58.33	7.93	0.63
One-shot	31.17	95.72	57.86	56.26	51.12	29.53	58.10	5.07	0.90
Few-shot	28.27	95.37	57.67	54.45	51.07	30.60	57.83	5.38	1.48
Chain-of-Thought	28.46	96.52	**58.33**	55.57	52.20	32.83	59.09	7.76	0.61
Tree-of-Thought	34.97	95.26	54.43	60.17	52.41	33.66	59.19	54.62	8.18
Chain-of-Table	**41.96**	92.47	53.47	61.53	50.70	32.63	58.16	**63.75**	**12.54**
Tree-of-Report	32.89	96.81	56.98	**63.33**	**54.92**	**36.37**	**61.68**	21.07	2.43

MLB	RG #	RG P%	CS P%	CS R%	CS F%	CO DLD%	Avg.	Time	Cost
Zero-shot	**53.17**	84.22	60.45	60.65	49.03	39.25	58.72	7.08	1.62
One-shot	41.15	90.91	70.94	61.95	56.67	47.32	65.56	8.72	1.77
Few-shot	44.16	89.69	71.64	61.91	56.68	46.80	65.34	10.36	1.78
Chain-of-Thought	39.88	94.11	73.72	63.40	56.79	46.46	66.90	9.34	1.73
Tree-of-Thought	33.78	95.83	73.28	**64.00**	59.34	48.50	68.19	50.96	7.25
Chain-of-Table	28.13	95.37	80.20	60.33	59.60	50.21	69.15	**55.60**	**10.80**
Tree-of-Report	30.78	**97.54**	**84.19**	63.48	**62.99**	**53.78**	**72.40**	29.18	6.77

ShuttleSet+	RG #	RG P%	CS P%	CS R%	CS F%	CO DLD%	Avg.	Time	Cost
Zero-shot	13.67	85.19	86.01	86.01	86.01	86.01	85.85	7.53	0.86
One-shot	12.22	84.02	83.26	74.42	78.38	56.99	75.42	6.59	1.12
Few-shot	14.33	90.22	87.72	86.58	86.99	82.31	86.76	6.00	2.20
Chain-of-Thought	13.67	85.53	84.97	84.62	84.70	83.49	84.66	6.68	0.81
Tree-of-Thought	13.33	81.92	81.35	82.48	81.88	81.35	81.80	63.11	9.62
Chain-of-Table	15.00	93.46	89.37	89.37	89.37	89.37	90.19	**73.67**	**14.44**
Tree-of-Report	**15.78**	**98.04**	**93.94**	**93.94**	**93.94**	**93.94**	**94.76**	29.04	5.71

Table 1: The quantitative results for RotoWire, MLB, and ShuttleSet+, respectively, where the highest scores are highlighted in **bold**, the second-highest scores are underlined, and our method is marked with a yellow background.

from $\hat{t}$ that also appear in t, indicating the information similarity between the generated text and the reference text. Content Ordering (CO) quantifies the complement of the Damerau-Levenshtein Distance (DLD%) (Damerau, 1964) between relations extracted from $\hat{t}$ and t, meaning the ordering similarity between the generated text and the reference text. We also compute the average (Avg.) of RG P%, CS P%, CS R%, CS F%, and CO DLD% to represent overall performance. Higher values of RG, CS, CO, and Avg. indicate better effectiveness.

Additionally, to evaluate the efficiency of each method, we compute the average time (in seconds) and cost (in $0.001 USD) required to generate a text. The cost is estimated based on the API Pricing published by OpenAI (OpenAI, 2025). Lower time and cost values mean better efficiency.

4.3 Implementation Detail

For all datasets, Tree-of-Report employs `gpt-4o-mini` (OpenAI, 2024) as the backbone LLM. Zero-shot prompting is used, with CSV as the table format, max degree set to 5, and the operation pool containing all available operations. However, since the texts in RotoWire and MLB are more general, we set the max depth to 3 for RotoWire and MLB. In contrast, because ShuttleSet+ contains more detailed texts, we set the max depth to 5 for ShuttleSet+. Additionally, we adjusted the prompts for Content Planning, `write()`, and Content Generating according to the requirements of each dataset.

4.4 Quantitative Result

We compared Tree-of-Report with other prompt-based methods on the RotoWire, MLB, and ShuttleSet+ datasets. The quantitative results are shown in Table 1.

From the quantitative results, we observe that Tree-of-Report achieves the best overall performance across all datasets, outperforming other baselines by 2.49% on RotoWire, 3.25% on MLB, and 4.57% on ShuttleSet+, demonstrating its superiority in effectiveness. This improvement is attributed to the tree-structured design of Tree-of-Report, which divides tables into smaller sub-tables and merges generated text into longer reports, facilitating high data fidelity and long-text generation.

Although Tree-of-Report does not achieve the

Prompt	RG #	RG P%	CS P%	CS R%	CS F%	CO DLD%	Time	Cost
Zero-shot	**15.78**	**98.04**	**93.94**	**93.94**	**93.94**	**93.94**	29.04	5.71
One-shot	15.67	97.04	93.77	93.77	93.77	93.77	**45.10**	15.03
Few-shot	14.78	95.99	93.86	89.41	91.08	87.93	42.88	**19.43**

Table 2: The experimental results for comparing different prompts on ShuttleSet+, where the highest scores are highlighted in **bold**, the second-highest scores are underlined, and the best configuration is marked with a yellow background.

Table Format	RG #	RG P%	CS P%	CS R%	CS F%	CO DLD%	Time	Cost
CSV	**15.78**	**98.04**	**93.94**	**93.94**	**93.94**	**93.94**	29.04	5.71
PIPE	**15.78**	**98.04**	93.29	93.29	93.29	93.29	78.53	9.63
HTML	15.67	97.39	92.64	92.64	92.64	92.64	**104.99**	**19.26**
Markdown	14.67	92.31	87.56	86.75	87.10	84.14	62.65	9.80

Table 3: The experimental results for comparing different table formats on ShuttleSet+, where the highest scores are highlighted in **bold**, the second-highest scores are underlined, and the best configuration is marked with a yellow background.

highest RG # on RotoWire and MLB, this is because the reports in these two datasets are more general. Compared to Zero-shot, which includes too little information, and Chain-of-Table, which includes too much information, Tree-of-Report selects an appropriate amount of information while maintaining a relatively high RG P%. This demonstrates the significance of Content Planning, which selects operations to extract relevant information.

Except for CS P% on RotoWire and CS R% on MLB, Tree-of-Report achieves the best performance in all CS metrics. The reason is that Tree-of-Report produces more detailed text, leading to higher scores on the detailed ShuttleSet+ dataset, but lower scores on the general RotoWire and MLB datasets. This reveals the significance of Operation Execution, which extracts more detailed information by executing operations to decompose the tables.

As for CO, Tree-of-Report achieves the highest score across all datasets. This shows the significance of Content Generating, which merges and rewrites texts to maintain the original structure and order of the tables.

Furthermore, while Tree-of-Report does not have the lowest time and cost, it is still lower than Tree-of-Thought and Chain-of-Table. For example, on ShuttleSet+, Tree-of-Report achieves only 39% of Chain-of-Table's time and 40% of its cost, showing its advantage in efficiency. This improvement is attributed to the optimizations that significantly reduce the time and cost of the Tree-of-Report.

The qualitative result is provided in Appendix D.

4.5 Ablation Study

4.5.1 The Effects of Prompt Templates

To analyze the impact of different prompts on Tree-of-Report, we design three types of prompts: Zero-shot, One-shot, and Few-shot. In the Zero-shot setting, the prompts for Content Planning, `write()`, and Content Generating do not include any human-annotated reference examples. The One-shot setting includes a single reference example in these prompts, while the Few-shot setting incorporates three reference examples. The experimental results on ShuttleSet+ are presented in Table 2.

The experimental results indicate that increasing the number of reference examples leads to worse performance while also increasing time and cost. Our assumption is that manually defined examples may constrain the LLM's inherent capabilities rather than enhance them. Additionally, the longer input context may make it more challenging for the LLM to identify the most relevant information. Consequently, we chose to use the Zero-shot prompt for all subsequent experiments.

4.5.2 The Impacts of Table Formats

We also analyzed the impact of different table formats on Tree-of-Report's performance by comparing four commonly used formats: CSV (Comma-Separated Values), PIPE, Markdown, and HTML (HyperText Markup Language). CSV separates values with commas, making it ideal for spreadsheets, while PIPE uses the `|` symbol as a delimiter, commonly found in command-line outputs. Markdown tables employ pipes (`|`) to define columns and hyphens (`-`) for headers, frequently used in documentation. HTML tables utilize `<table>`, `<tr>`, and `<td>` tags to structure tabular data for the website.

The experimental results on ShuttleSet+ in Table 3 show that CSV achieves the best performance. While PIPE and HTML perform similarly, they have significantly higher time and cost due to requiring more symbols to represent the table, resulting in a longer input context. Markdown performs the worst, likely because LLMs have been pre-trained on fewer examples of this format, leading to weaker table comprehension. Based on these findings, we adopted CSV as the table format for all following experiments.

Max Depth	Max Degree	RG #	RG P%	CS P%	CS R%	CS F%	CO DLD%	Time	Cost
5	5	**15.78**	**98.04**	**93.94**	**93.94**	**93.94**	**93.94**	29.04	**5.71**
3	5	15.67	95.99	91.42	92.72	92.03	91.42	16.60	2.37
5	3	15.67	97.21	92.46	92.46	92.46	92.46	**29.48**	4.90
3	3	13.89	88.18	83.43	82.00	82.56	82.00	11.79	1.82

Table 4: The experimental results for comparing different max depth and max degree on ShuttleSet+, where the highest scores are highlighted in **bold**, the second-highest scores are underlined, and the best configuration is marked with a yellow background.

Operation Pool	RG #	RG P%	CS P%	CS R%	CS F%	CO DLD%	Time	Cost
All operations	**15.78**	98.04	**93.94**	93.94	**93.94**	**93.94**	29.04	5.71
w/o `select_table()`	15.44	**98.69**	82.57	92.94	85.57	82.57	44.45	6.11
w/o `select_row()`	15.33	98.04	84.53	94.90	87.53	84.53	48.00	6.80
w/o `select_col()`	15.11	**98.69**	85.19	93.33	86.95	82.96	**49.80**	**7.49**
w/o `count()`	15.44	**98.69**	82.57	92.94	85.57	82.57	25.30	4.20
w/o `sort()`	15.44	**98.69**	85.19	**95.56**	88.18	85.19	36.64	5.64
w/o `filter()`	15.44	**98.69**	82.57	92.94	85.57	82.57	33.34	5.53

Table 5: The experimental results for comparing different operation pools on ShuttleSet+, where the highest scores are highlighted in **bold**, the second-highest scores are underlined, and the best configuration is marked with a yellow background.

4.5.3 The Analysis of Max Depth & Max Degree

To determine the optimal maximum depth and maximum degree for Tree-of-Report, we conducted the following experiments. The baseline setting uses a max depth and max degree of 5. In one experiment, we reduced the max depth to 3 while keeping the max degree at 5. In another, we set the max degree to 3 while maintaining the max depth at 5. Finally, we tested a configuration where both the max depth and max degree were set to 3. The experimental results on ShuttleSet+ are presented in Table 4.

From the experimental results, we observe that setting both max depth and max degree to 5 yields the best performance; however, it also results in higher time and cost. When reducing the max depth to 3 while keeping the max degree at 5, the performance drops more significantly compared to reducing the max degree to 3 while keeping the max depth at 5. This suggests that max depth has a greater impact on the generated text than max degree. This finding is intuitive, as max depth controls the level of detail in the text, whereas max degree influences its richness.

Finally, setting both max depth and max degree to 3 yields the worst performance, as expected. However, it is worth noting that this setting also results in the lowest time and cost. This suggests that max depth and max degree can be adjusted based on the desired level of detail in the generated text. If more detailed text is required, increasing max depth and max degree improves performance at the expense of higher computational cost. Conversely, for more general text, reducing the max depth and max degree lowers both the level of detail and the cost.

4.5.4 The Influences of Operation Pool

To demonstrate the significance of each operation, we performed the following experiments. The baseline configuration includes all operations in the operation pool. Then, in each experiment, we systematically removed one operation from the operation pool and evaluated the impact on performance.

The experimental results on ShuttleSet+ in Table 5 show that removing `select_table()`, `select_row()`, and `select_col()` leads to a significant performance drop, highlighting their importance. Without these operations, the LLM processes the entire table to generate text, leading to increased time and cost. In contrast, removing `count()`, `sort()`, and `filter()` has a less pronounced effect, suggesting that they are relatively less critical. However, without these operations, it becomes impossible to compute more detailed information, resulting in a lower RG # but a higher RG P%. Overall, maintaining all operations provides the most balanced performance, demonstrating greater robustness.

5 Conclusion

In this paper, we propose Tree-of-Report, a novel framework for table-to-text generation in sports game reports. Inspired by the "divide and conquer" concept of merge sort (Bron, 1972), our method divides the generation process into three stages: Content Planning, Operation Execution, and Content Generating. By recursively decomposing large tables into smaller sub-tables and merging short texts into a long text, our approach effectively enhances data fidelity and generates coherent long-form outputs. Experimental results demonstrate that Tree-of-Report achieves the best overall performance across all datasets, surpassing other prompt-based baselines by 2.49% on RotoWire, 3.25% on MLB, and 4.57% on ShuttleSet+, highlighting its superiority in effectiveness. Furthermore, our method achieves only 39% of Chain-of-Table's time and 40% of its cost, showing its advantage in efficiency. In summary, Tree-of-Report opens a new path for prompt-based table-to-text generation in sports game reports.

Limitations

While Tree-of-Report demonstrates strong performance, our approach requires manually tuning configurations and prompts for the corresponding dataset. Therefore, one of the interesting research directions could be to explore automatic selection for configurations and prompts. On the other hand, Tree-of-Report achieves more efficient time and cost compared to Tree-of-Thought and Chain-of-Table, yet we leave the efficiency direction as the future work as our proposed approach still requires higher time and cost than Few-shot and Chain-of-Thought.

Ethical Considerations

First, although Tree-of-Report does not require large training datasets compared to model-based baselines, using external LLMs raises concerns about data privacy, especially for sensitive information. Second, while Tree-of-Report achieves higher data fidelity compared to other prompt-based baselines, hallucination issues may still occur, potentially generating incorrect or non-existent information that could mislead readers.

Acknowledgments

In this work, we used Copilot to automatically complete some basic code and utilized ChatGPT for grammar corrections, language translation, and literature searches. All uses were reviewed to ensure compliance with the ACL Rolling Review's AI Writing/Coding Assistance Policy.

References

C. Bron. 1972. Algorithm 426: Merge sort algorithm [m1]. *Commun. ACM*, 15(5):357–358.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Kai Chen, Fayuan Li, Baotian Hu, Weihua Peng, Qingcai Chen, Hong Yu, and Yang Xiang. 2021. Neural data-to-text generation with dynamic content planning. *Knowledge-Based Systems*, 215:106610.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact: A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations*.

Frederick J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176.

Heng Gong, Wei Bi, Xiaocheng Feng, Bing Qin, Xiaojiang Liu, and Ting Liu. 2020. Enhancing content planning for table-to-text generation with data understanding and verification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2905–2914, Online. Association for Computational Linguistics.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213.

Yupian Lin, Tong Ruan, Jingping Liu, and Haofen Wang. 2024. A survey on neural data-to-text generation. *IEEE Transactions on Knowledge and Data Engineering*, 36(4):1431–1449.

OpenAI. 2024. Gpt-4o mini: Advancing cost-efficient intelligence.

OpenAI. 2025. API Pricing. Accessed: 2025-02-15.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. Totto: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019a. Data-to-text generation with content selection and planning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6908–6915.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019b. Data-to-text generation with entity modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2023–2035, Florence, Italy. Association for Computational Linguistics.

Ratish Puduppully, Yao Fu, and Mirella Lapata. 2022. Data-to-text generation with variational sequential planning. *Transactions of the Association for Computational Linguistics*, 10:697–715.

Ratish Puduppully and Mirella Lapata. 2021. Data-to-text generation with macro planning. *Transactions of the Association for Computational Linguistics*, 9:510–527.

Wei-Yao Wang, Wei-Wei Du, and Wen-Chih Peng. 2023. Shuttleset22: Benchmarking stroke forecasting with stroke-level badminton dataset. *CoRR*, abs/2306.15664.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. Chain-of-table: Evolving

tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.

A Example Prompt

A.1 Example Prompt for Content Planning

Figure 3 shows an example prompt for Content Planning on the ShuttleSet+ dataset. In this prompt, `{TABLE_DESCRIPTION}` provides descriptions for each column in the table, `{OPERATION_DESCRIPTION}` explains each available operation, `{TABLES}` represents the input tables, `{OPERATION_HISTORY}` lists the operations previously used by the parent node, and `{OPERATION_POOL}` indicates the remaining unused operations. Then, the LLM outputs the `Operations & Arguments` based on the input prompt.

A.2 Example Prompt for `write()` operation

Figure 4 shows an example prompt for the `write()` operation on the ShuttleSet+ dataset. In this prompt, `{TABLE_DESCRIPTION}` provides descriptions for each column in the table, and `{TABLES}` represents the input tables. Then, the LLM generates the `Report` based on the input prompt.

A.3 Example Prompt for Content Generating

Figure 5 shows an example prompt for Content Generating on the ShuttleSet+ dataset. In this prompt, `{REPORTS}` represents multiple reports. Then, the LLM merges and rewrites the reports into a `New Report` based on the input prompt.

A.4 Example Prompt for the LLM-based IE model

Figure 6 shows an example prompt for the LLM-based IE model on the ShuttleSet+ dataset. In this prompt, `{TABLE_DESCRIPTION}` provides descriptions for each column in the table, `{REPORT}` is a report of a match, and `{TABLE_RELATION}` lists all relations from the match's tables. The LLM then extracts relations from the report based on the input prompt.

B Data Preprocessing

B.1 Data Preprocessing for RotoWire

To ensure the input aligns with the Tree-of-Report format, we first preprocess the RotoWire dataset. Initially, we convert the original data from JSON format into multiple CSV tables: `game`, `home_line`, `vis_line`, and `box_score`. Specifically, `game` contains overall game information, `home_line` represents the line score of the home team, `vis_line` represents the line score of the visiting team, and `box_score` records individual player statistics. Finally, we reorder the table columns according to the sequence specified in the table description of RotoWire.

B.2 Data Preprocessing for MLB

To ensure the input conforms to the Tree-of-Report format, we also preprocess the MLB dataset. Similar to RotoWire, we first convert the original data from JSON format into multiple CSV tables: `game`, `home_line`, `vis_line`, `box_score`, and `play_by_play`. Specifically, `game` contains overall game information, `home_line` represents the line score of the home team, `vis_line` represents the line score of the visiting team, `box_score` records individual player statistics, and `play_by_play` details the scoring of each at-bat. Next, since the `box_score` table contains many rows with `N/A` values, we remove these redundant rows to streamline the dataset. Eventually, we reorder the table columns according to the sequence specified in the table description of MLB.

B.3 Data Preprocessing for ShuttleSet+

ShuttleSet22 is a stroke-level dataset; however, generating textual descriptions does not require such detailed information. Therefore, we retain only the final stroke of each rally. To streamline the dataset, we selected the nine most essential columns, renaming and reordering to improve clarity while

Prompt	RG #	RG P%	CS P%	CS R%	CS F%	CO DLD%
Zero-shot	**15.00**	**100.00**	73.33	**64.71**	68.75	17.65
One-shot	11.00	**100.00**	**100.00**	**64.71**	**78.57**	**64.71**

Table 6: The evaluation results of the LLM-based IE model with different prompts. The highest scores are highlighted in **bold**, and the best configuration is marked with a yellow background.

removing unrelated fields. Additionally, the values in the `ball_type`, `win_reason`, and `lose_reason` columns were originally in Chinese, so we translated them into English. Lastly, we reorder the table columns according to the order specified in the table description of ShuttleSet+.

C LLM-based IE model

Since no existing IE model is available for ShuttleSet+ and the training data are insufficient, we use an LLM as the IE model to extract relations from the text. To validate the reliability of the LLM-based IE model, we manually annotated a set of relations and compared them with those extracted by the LLM. The evaluation results are presented in Table 6.

We compared two prompting methods for the LLM-based IE model: Zero-shot and One-shot. Experimental results show that One-shot performs better, achieving over 60% across all metrics. Therefore, we used One-shot prompting for all subsequent experiments. We hypothesize that providing an additional example allows the LLM to reference it, leading to relation extraction that more closely aligns with human annotations. The prompt for the LLM-based IE model is provided in Appendix A.4.

D Qualitative Result

Figure 7 showcases the qualitative results of human-written, Chain-of-Table, and Tree-of-Report outputs. For ease of comparison, we mark the information in the text with **bold**: green indicates information included in the tables, while red indicates errors or information not found in the tables.

From the qualitative results, we observe that compared to Chain-of-Table, Tree-of-Report generates more comprehensive and detailed information (e.g., shot type frequencies) and produces more accurate outputs, with only one error compared to seven errors from Chain-of-Table. This further validates that Tree-of-Report can generate text that meets the characteristics of high data fidelity and long-form output for sports game reports.

Algorithm 1 Tree-of-Report

Require: Tables T, Operation History OH, Operation Pool OP, Level L, Max Depth MAX_DEPTH, Max Degree MAX_DEGREE
Ensure: Text t

function TREE-OF-REPORT(T, OH, OP, L)
  if $L \geq MAX\_DEPTH$ **then**
    $t \leftarrow$ WRITE(T)
    return t
  end if
  ▷ Content Planning
  $OA \leftarrow$ CONTENT_PLANNING(T, OH, OP)
  ▷ Operation Execution
  $t' \leftarrow ()$
  for each (O_i, A_i) in $OA[0 : MAX\_DEGREE]$ **do**
    if $O_i = write()$ **then**
      $t'_i \leftarrow$ WRITE(T)
    else
      $T_i \leftarrow O_i(T, A_i)$
      $OH_i \leftarrow OH + O_i(A_i)$
      $OP_i \leftarrow OP - O_i()$
      $L_i \leftarrow L + 1$
      $t'_i \leftarrow$ TREE-OF-REPORT(T_i, OH_i, OP_i, L_i)
    end if
    $t' \leftarrow t' + t'_i$
  end for
  ▷ Content Generating
  $t \leftarrow$ CONTENT_GENERATING(t')
  return t
end function

Main Program
$T \leftarrow (T^j \mid j = 1, 2, \ldots, n)$
$OH \leftarrow (op \mid op = \text{root}())$
$OP \leftarrow (op \mid op \in \text{operations},\ op \neq \text{root}())$
$L \leftarrow 0$
$t \leftarrow$ TREE-OF-REPORT(T, OH, OP, L)

```
System:

You are a content planner for the badminton game report.

Please select candidate Operations and corresponding Arguments from the Operation
    Pool based on the input Tables and Operation History. These candidate Operations
     will be the next Operation in the Operation History.

# Requirements

1. Strictly adhere to the requirements.
2. The output must be in English.
3. The output must be based on the input data; do not hallucinate.
4. The table format is {TABLE_FORMAT}.
5. The length of Operation History must be less than or equal to {MAX_DEPTH}.
6. The number of Operations must be less than or equal to {MAX_DEGREE}.
7. Only select Operations from the Operation Pool.
8. Arguments must match the format required by the corresponding Operations.
9. Operations & Arguments must follow this format: [operation_1(argument_1, ...),
    operation_2(argument_2, ...), operation_3(argument_3, ...), ...]
10. Only output Operations & Arguments!
11. The number of tokens in the Operations & Arguments must be within {
    PLANNING_TOKENS}.

# Table Description

{TABLE_DESCRIPTION}

# Operation Description

{OPERATION_DESCRIPTION}

User:

# Test

## Tables

{TABLES}

## Operation History

{OPERATION_HISTORY}

## Operation Pool

{OPERATION_POOL}

## Operations & Arguments
```

Figure 3: Prompt for Content Planning

```
System:

You are a content writer for the badminton game report.

Please write the Report based on the input Table.

# Requirements

1. Strictly adhere to the requirements.
2. The output must be in English.
3. The output must be based on the input data; do not hallucinate.
4. The Table format is {TABLE_FORMAT}.
5. The Report can only describe the content included in the Tables and cannot
    describe anything not included in the Tables.
6. The Report must consist of only one paragraph.
7. The number of tokens in the Report must be within {WRITE_TOKENS}.

# Table Description

{TABLE_DESCRIPTION}

User:

# Test

## Tables

{TABLES}

## Report
```

Figure 4: Prompt for `write()` operation

```
System:

You are a content generator for the badminton game report.

Please merge and rewrite a New Report based on the input Reports.

# Requirements

1. Strictly adhere to the requirements.
2. The output must be in English.
3. The output must be based on the input data; do not hallucinate.
4. The New Report must include all the content from the input Reports; do not omit
    any information.
5. The New Report must follow the order of the input Reports.
6. The number of tokens in the New Report must be within {GENERATING_TOKENS}.

User:

# Test

## Reports

{REPORTS}

## New Report
```

Figure 5: Prompt for Content Generating

```
System:

You are a relation extractor for the badminton game report.

Please extract the Report Relation contained in the Report from the Table Relation.

There is an Example that you can refer to.

# Requirements

1. Strictly adhere to the requirements.
2. The output must be in English.
3. The output must be based on the input data; do not hallucinate.
4. Please do not output any Report Relation that is not included in the Report.
5. Please do not output any Report Relation that is not included in the Table
    Relation.
6. The Report Relation must contain all the relations from the input Report; do not
    omit any relation.
7. The Report Relation must follow the order in the input Report.
8. The Report Relation must follow the format: [(table|column|value), (table|column|
    value), ...]

# Table Description

{TABLE_DESCRIPTION}

User:

# Test

## Report

{REPORT}

## Table Relation

{TABLE_RELATION}

## Report Relation
```

Figure 6: Prompt for LLM-based IE model

Human

Yamaguchi Akane defeats **Chen Yufei** in the women's singles **semi-final**

Yamaguchi Akane has beaten **Chen Yufei 21-11**, **21-13** in the women's **All England semi-final**, setting up a final with An Seyoung tomorrow, Sunday 20 March.

Billed as a battle between the world champ and the Olympic champ, **Yamaguchi** came out on top and put on a clinic after a slow start.

She came from **1-5** down to clinch the **first game 21-11** and never looked back, **Chen** simply had no answer to **Yamaguchi**'s all-action style as she returned absolutely everything and took her chances clinically.

11-4 ahead at the interval of **game two** there was no coming back for **Chen** and **Yamaguchi** put it away with some breathtaking badminton.

She'll face South Korean An tomorrow who also had a straight games victory over Tai Tzu Ying in her semi-final.

Chain-of-Table

In an exhilarating badminton match, **Akane Yamaguchi** faced off against **Chen Yufei** in a thrilling contest that showcased remarkable skill and determination from both players. The match unfolded over **two sets**, with each player demonstrating their prowess on the court.

In the **first set**, **Yamaguchi** started strong, winning the first rally with a **lob** that **went out of bounds for her opponent**. However, **Chen** quickly responded, taking the next rally with a **net shot** that also **went out of bounds**. The set saw a series of intense rallies, with **Yamaguchi** employing a mix of **net shots** and **smashes**, while **Chen** countered with strategic **clears** and **lobs**. The **first set** concluded with **Yamaguchi** edging out **Chen**, winning **12** points to Chen's **8**.

The **second set** continued the fierce competition, with both players exchanging points in rapid succession. **Yamaguchi**'s agility and tactical play were on full display as she executed a series of successful **smashes** and **defensive returns**. **Chen**, not to be outdone, showcased her own skills with powerful **wrist smashes** and precise **clears**. The set was marked by several **long rallies**, with both players refusing to give an inch.

Ultimately, **Yamaguchi** emerged victorious, winning the match with a final score of **21-17** in the **second set**. The match highlighted the exceptional talent of both players, with **Yamaguchi**'s ability to adapt her strategy and **Chen**'s relentless fighting spirit making for an unforgettable encounter. Fans were treated to a display of top-tier badminton, leaving them eagerly anticipating their next showdown.

Tree-of-Report

In the **semi-finals** of the **YONEX All England Open Badminton Championships 2022**, **Akane Yamaguchi** faced off against **CHEN Yufei** in a thrilling match that lasted **41 minutes**. **Yamaguchi** emerged victorious, winning in **two sets** with scores of **21-11** and **21-13**.

In the **first set**, both players showcased their skills, with **CHEN Yufei** initially taking the lead. **CHEN** displayed impressive shots, including a successful **lob** that forced **Akane out of bounds** and a decisive **smash**. However, **Akane Yamaguchi** demonstrated her dominance by winning a total of **16 rallies**, showcasing her exceptional skills and strategic play. She capitalized on **CHEN**'s errors, including **landing judgment mistakes** and **hitting the net**, effectively turning the tide in her favor. The set concluded with **Yamaguchi** scoring **21** points to **CHEN**'s **11**.

The **first set** featured a diverse range of shot types, with "**return net**" being the most frequent at **7** occurrences, followed by "**lob**" at **6** and "**clear**" at **4**. Other notable shots included "**drop**" with **3**, "**smash**" with **2**, and **single** instances of "**cross-court net shot**," "**net shot**," and "**rush**." This variety contributed to the dynamics of the set.

In the **second set**, **Akane Yamaguchi** continued her strong performance, winning **21 rallies** compared to **CHEN Yufei**'s **13**. **Yamaguchi** utilized a series of effective shots, including a **lob** and a **back-court drive**, while **CHEN** managed to respond with a **smash** and a **wrist smash**, winning some points. The **second set** was marked by strategic plays and errors from both players, but **Yamaguchi** maintained her dominance, ultimately winning the set **21-13**.

The **second set** showcased a different shot distribution, with the **smash** being the most frequent, occurring **7** times. The **wrist smash** followed closely with **4** instances, while both the **return net** and lob were executed **4** and **3** times, respectively. Other notable shots included the **lob** and **net shot**, each appearing **3** times, along with **2 defensive return lobs** and a **back-court drive**, highlighting the diverse range of techniques employed by both players.

Overall, **Akane Yamaguchi**'s performance in the **semi-finals** of the **YONEX All England Open Badminton Championships** was a testament to her skill and strategic gameplay, leading her to a well-deserved victory against **CHEN Yufei**.

Figure 7: The qualitative results of human-written, Chain-of-Table, and Tree-of-Report outputs.