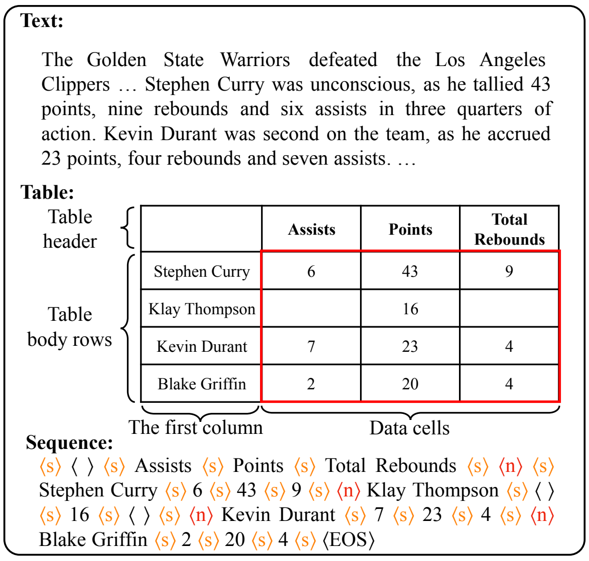
A Sequence-to-Sequence&Set Model for Text-to-Table Generation

Tong Li[1,2*] **Zhihao Wang**[1,2*] **Liangying Shao**[1] **Xuling Zheng**[1,2†]
Xiaoli Wang[1] **Jinsong Su**[1,2†]
[1]School of Informatics, Xiamen University, China
[2]Key Laboratory of Digital Protection and Intelligent Processing of Intangible Cultural Heritage of Fujian and Taiwan (Xiamen University), Ministry of Culture and Tourism, China
{litong, zhwang, liangyingshao}@stu.xmu.edu.cn {xlzheng, xlwang, jssu}@xmu.edu.cn

Abstract

Recently, the text-to-table generation task has attracted increasing attention due to its wide applications. In this aspect, the dominant model (Wu et al., 2022) formalizes this task as a sequence-to-sequence generation task and serializes each table into a token sequence during training by concatenating all rows in a top-down order. However, it suffers from two serious defects: 1) the predefined order introduces a wrong bias during training, which highly penalizes shifts in the order between rows; 2) the error propagation problem becomes serious when the model outputs a long token sequence. In this paper, we first conduct a preliminary study to demonstrate the generation of most rows is order-insensitive. Furthermore, we propose a novel sequence-to-sequence&set text-to-table generation model. Specifically, in addition to a *text encoder* encoding the input text, our model is equipped with a *table header generator* to first output a table header, i.e., the first row of the table, in the manner of sequence generation. Then we use a *table body generator* with learnable row embeddings and column embeddings to generate a set of table body rows in parallel. Particularly, to deal with the issue that there is no correspondence between each generated table body row and target during training, we propose a target assignment strategy based on the bipartite matching between the first cells of generated table body rows and targets. Experiment results show that our model significantly surpasses the baselines, achieving state-of-the-art performance on commonly-used datasets.[1]



Text:

The Golden State Warriors defeated the Los Angeles Clippers … Stephen Curry was unconscious, as he tallied 43 points, nine rebounds and six assists in three quarters of action. Kevin Durant was second on the team, as he accrued 23 points, four rebounds and seven assists. …

Table:

	Assists	**Points**	**Total Rebounds**
Stephen Curry	6	43	9
Klay Thompson		16	
Kevin Durant	7	23	4
Blake Griffin	2	20	4

Sequence:

⟨s⟩ ⟨ ⟩ ⟨s⟩ Assists ⟨s⟩ Points ⟨s⟩ Total Rebounds ⟨s⟩ ⟨n⟩ ⟨s⟩ Stephen Curry ⟨s⟩ 6 ⟨s⟩ 43 ⟨s⟩ 9 ⟨s⟩ ⟨n⟩ Klay Thompson ⟨s⟩ ⟨ ⟩ ⟨s⟩ 16 ⟨s⟩ ⟨ ⟩ ⟨s⟩ ⟨n⟩ Kevin Durant ⟨s⟩ 7 ⟨s⟩ 23 ⟨s⟩ 4 ⟨s⟩ ⟨n⟩ Blake Griffin ⟨s⟩ 2 ⟨s⟩ 20 ⟨s⟩ 4 ⟨s⟩ ⟨EOS⟩

Figure 1: An example of text-to-table task. The input text is a report of a basketball game. In the existing dominant model (Wu et al., 2022), the output table is serialized into a token sequence during training by concatenating all rows in a top-down order. Here, $\langle s \rangle$ token is used to separate the cells of each row, $\langle n \rangle$ token is utilized to separate rows, and ⟨ ⟩ token means an empty cell. Unlike Wu et al. (2022), in this work, we model the generation of each table as a table header and then a set of table body rows. Note that these rows can be further decomposed into the first column and data cells wrapped in the red box.

1 Introduction

Text-to-table generation is a task that aims to generate a tabular description of important information for a given text. As shown in Figure 1, the input text is a post-game summary of a basketball game, and the output is a table containing statistics about players. Usually, this task can be widely used to extract important structured information, such as restaurant reviews (Novikova et al., 2017), team and player statistics (Wiseman et al., 2017), Wikipedia infoboxes (Bao et al., 2018) and biographies (Lebret et al., 2016), benefiting humans understand the input text more intuitively.

In this aspect, Wu et al. (2022) first propose sequence-to-sequence (seq2seq) text-to-table generation models. They first serialize each table to a

[1]We release the code at https://github.com/DeepLearnXMU/seq2seqset
[*]Equal contribution.
[†]Corresponding author.

token sequence by concatenating all rows in a top-down order. Back to Figure 1, they represent each table row as a cell sequence, and then represent the entire table by concatenating all rows. Then, they train seq2seq models fine-tuned from BART (Lewis et al., 2020). During inference, the model generates a table in a token-by-token manner and the generated sequence is eventually split by $\langle s \rangle$ and $\langle n \rangle$ to obtain the structured table.

Despite some success, the above-mentioned sequence generation manner leads to two defects in the model. **First**, imposing the above-mentioned predefined order on generating rows in the dataset may bring wrong bias to the model training (Ye et al., 2021; Lu et al., 2022a). Here, we still take Figure 1 as an example. Each row represents the statistics of a basketball player, and there is no obvious dependency between the statistics of different players. Thus, when considering the order of generating rows, the inconsistent order between generated rows and targets will cause a large training loss, even if the generated rows and targets are exactly the same. **Second**, as the number of generated rows increases, the outputted token sequence becomes longer, which makes the seq2seq models encounter the serious error propagation problem (Ye et al., 2021; Tan et al., 2021). Besides, the seq2seq model generates a table autoregressively, of which time complexity is the number of rows times the number of columns, resulting in inefficient GPU acceleration.

In this paper, we first conduct a preliminary study to inspect the effect of row generation order on seq2seq models. Specifically, we randomly reorder table body rows to construct different training datasets. Then, we use these datasets to train seq2seq models, of which performance is compared on the same dataset. Experimental results show that these models exhibit similar performance, proving that the generation of most table body rows is order-insensitive.

Moreover, we propose a novel sequence-to-sequence&set (Seq2Seq&set) text-to-table generation model which decomposes the table generation into two steps: generating a table header, i.e., the first row of the table, and then a set of table body rows. As shown in Figure 2, our model mainly consists of three modules: 1) *Text Encoder*. It is a vanilla Transformer encoder, encoding the input document into hidden states; 2) *Table Header Generator* that produces the table header as a token sequence; 3) *Table Body Generator* generating different table body rows in parallel, where each row is generated token by token. To generate different rows from the same text, we equip the generators with a set of learnable *Row Embeddings*. Besides, we add a set of learnable *Column Embeddings* to enhance the semantic consistency between cells in the same column.

During the model training, we need to determine the correspondence between the generated table body rows and targets, so as to achieve the order-independent generation of table body rows. To this end, we propose to use the model to generate the first cells of table body rows. Then, we efficiently determine target assignments according to the matching results between these first cells and those of targets, which can be modeled as a bipartite matching problem and solved by the Hungarian algorithm (Kuhn, 1955). Afterwards, we calculate the training loss based on the one-to-one alignments between the generated table body rows and targets. Besides, during the model inference, we force table body generator to output the same number of cells with the previously-generated generated table header.

Compared with the seq2seq models (Wu et al., 2022), our model has the following advantages: 1) our model is able to not only alleviate the order bias caused by the sequence generation but also reduce the effect of error propagation on the generation of long sequences; 2) our model achieves faster generation speed since table body rows can be efficiently generated in parallel.

Experiment results show that our model significantly improves the quality of the generated table, achieving state-of-the-art performance on commonly-used datasets.

2 Related Work

Information Extraction (IE) refers to the automatic extraction of structured information such as entities, relationships between entities, and attributes describing entities from unstructured sources (Sarawagi et al., 2008). The common IE tasks include named entity recognition (NER), relation extraction (RE), event extraction (EE), etc.

To achieve high-quality IE, researchers have proposed various task-specific IE methods. With the development of deep learning, researchers mainly focus on neural network based generation models, which are often seq2seq pre-trained models generating serialized structured information. Compared with traditional IE methods, these methods have achieved comparable or even superior results in RE (Zeng et al., 2018; Nayak and Ng, 2020), NER (Chen and Moschitti, 2018; Yan et al., 2021), EE (Li et al., 2021; Lu et al., 2021). Along this line, researchers resort to unified models (Paolini et al., 2021; Lu et al., 2022b) that model multiple IE tasks as the generation of sequences in a uniform format.

In this work, we mainly focus on text-to-table generation that aims to generate structured tables from natural language descriptions. Note that text-to-table generation can be considered as the dual task of table-to-text generation, which intends to generate a textual description conditioned on the input structured data. Usually, these structured data are represented as tables (Wiseman et al., 2017; Thomson et al., 2020; Chen et al., 2020) or sets of table cells (Bao et al., 2018; Parikh et al., 2020). Compared with traditional IE tasks, this task does not rely on predefined schemas. In this regard, Wu et al. (2022) first explore this task as a seq2seq generation task by fine-tuning a BART model (Lewis et al., 2020) for generation. By contrast, we model the generation of each table as a table header and then a set of table body rows. To this end, we propose a Seq2Seq&set text-to-table model, which can alleviate the defects caused by the sequence generation in the conventional seq2seq models.

3 Preliminary Study

We first conduct a preliminary study to inspect the effect of row generation order on the Seq2Seq model (Wu et al., 2022). Since the table header is the first row of a table containing column names which should be generated first, so we only randomly reorder table body rows to construct different training datasets. Then, we individually train the Seq2Seq models using these datasets with the same setting as the original model, and compare their performance on the same dataset (Wiseman et al., 2017).

From Table 1, we can observe that the original model and their variants exhibit similar performance. Besides, we calculate the sample standard deviation of model performance and find that all standard deviations are no more than 0.1. These results strongly demonstrate that the generation orders of table body rows have a negligible effect on the model performance. In other words, the generation of table body rows is order-insensitive.

Subset	Model	The first column F1	Table header F1	Data cell F1
Team	Origin	94.71	86.07	82.97
	Random1	94.57	85.93	82.83
	Random2	94.76	86.01	83.00
	Random3	94.66	85.91	82.84
	S_{Team}	**0.08**	**0.07**	**0.09**
Player	Origin	92.16	87.82	81.96
	Random1	92.23	87.66	81.79
	Random2	92.33	87.69	81.84
	Random3	92.13	87.85	81.99
	S_{Player}	**0.09**	**0.07**	**0.10**

Table 1: Results of preliminary study on the Team and Player subsets in the Rotowire dataset. We calculate exact match F1 scores on three table parts respectively. "*Origin*" means the Seq2Seq model trained using the original dataset, and "*Random1-3*" denote models trained on three datasets with different table body row orders, respectively. S_* means the sample standard deviation of the model performance. [2]

4 Our Model

In this section, we describe our model in detail. As shown in Figure 2, our model is composed of three modules: *Text Encoder*, *Table Header Generator* and *Table Body Generator*. Then, we give a detailed description of the model training.

4.1 Text Encoder

Our text encoder is used to encode input documents. It is identical to the BART (Lewis et al., 2020) encoder, consisting of L_e Transformer encoder layers. The input document is first tokenized into X=$x_1, x_2, ..., x_{|\text{X}|}$ using a byte-level Byte-Pair Encoding (Wang et al., 2020) tokenizer.

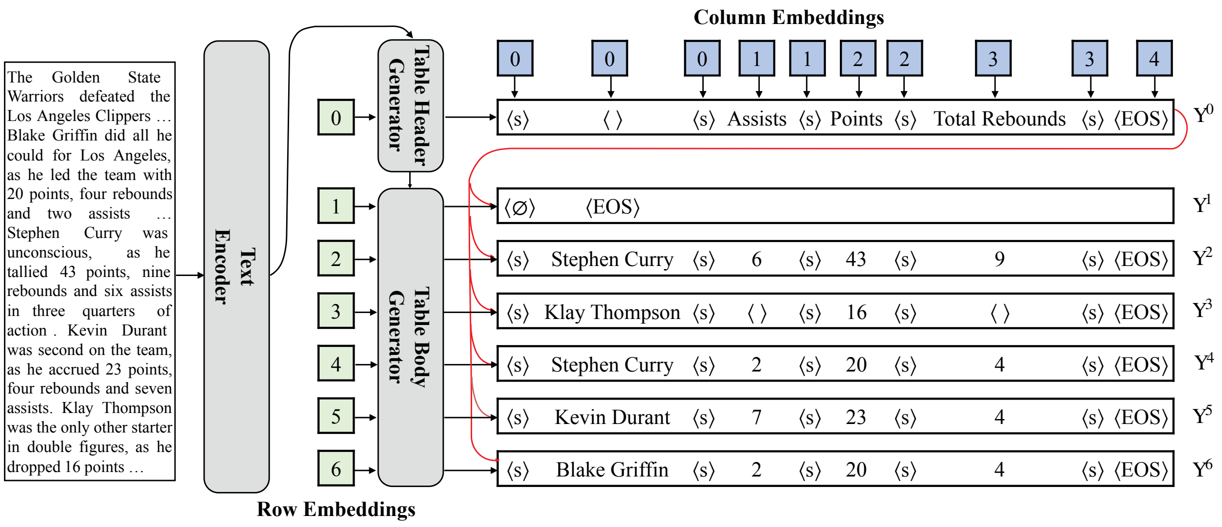
Then, text encoder iteratively updates the hidden states in the following way:

$$\mathbf{A}_e^{(l)} = \text{MultiHead}(\mathbf{H}_e^{(l)}, \mathbf{H}_e^{(l)}, \mathbf{H}_e^{(l)}), \quad (1)$$

$$\mathbf{H}_e^{(l+1)} = \text{FFN}(\mathbf{A}_e^{(l)}), \quad (2)$$

where $\mathbf{H}_e^{(l)} \in \mathbb{R}^{|\text{X}| \times d}$ is the hidden states at the l-th layer, and d is the dimension of embeddings and hidden states. $\text{MultiHead}(\cdot)$ is a multi-head attention function and $\text{FFN}(\cdot)$ refers to a feed-forward network. We initialize $\mathbf{H}_e^{(0)}$ as the sum of $\text{Word}(\text{X})$ and $\text{Pos}(\text{X})$, where $\text{Word}(\cdot)$ is a word embedding

[2]Notice that in (Wu et al., 2022), they use *row header F1*, *column header F1* and *non-header cells F1*. Here, we individually rename these to *the first column F1*, *table header F1* and *data cell F1*, so as to avoid ambiguity in descriptions.



Column Embeddings
0 0 0 1 1 2 2 3 3 4
⟨s⟩ ⟨ ⟩ ⟨s⟩ Assists ⟨s⟩ Points ⟨s⟩ Total Rebounds ⟨s⟩ ⟨EOS⟩ Y0
⟨∅⟩ ⟨EOS⟩ Y1
⟨s⟩ Stephen Curry ⟨s⟩ 6 ⟨s⟩ 43 ⟨s⟩ 9 ⟨s⟩ ⟨EOS⟩ Y2
⟨s⟩ Klay Thompson ⟨s⟩ ⟨ ⟩ ⟨s⟩ 16 ⟨s⟩ ⟨ ⟩ ⟨s⟩ ⟨EOS⟩ Y3
⟨s⟩ Stephen Curry ⟨s⟩ 2 ⟨s⟩ 20 ⟨s⟩ 4 ⟨s⟩ ⟨EOS⟩ Y4
⟨s⟩ Kevin Durant ⟨s⟩ 7 ⟨s⟩ 23 ⟨s⟩ 4 ⟨s⟩ ⟨EOS⟩ Y5
⟨s⟩ Blake Griffin ⟨s⟩ 2 ⟨s⟩ 20 ⟨s⟩ 4 ⟨s⟩ ⟨EOS⟩ Y6
Table Header Generator
Table Body Generator
Text Encoder
0 1 2 3 4 5 6
Row Embeddings
The Golden State Warriors defeated the Los Angeles Clippers … Blake Griffin did all he could for Los Angeles, as he led the team with 20 points, four rebounds and two assists … Stephen Curry was unconscious, as he tallied 43 points, nine rebounds and six assists in three quarters of action . Kevin Durant was second on the team, as he accrued 23 points, four rebounds and seven assists. Klay Thompson was the only other starter in double figures, as he dropped 16 points …

Figure 2: The architecture of our model. It mainly consists of three modules: text encoder encoding input text, table header generator producing a table header as a sequence, and table body generator that generates a set of table body rows in parallel. Note that we have two kinds of learnable embeddings: row embeddings (green squares) enabling table body generator to produce different table body rows, and column embeddings (blue squares) used to enhance the semantic consistency between cells in the same column.

function and $\mathrm{Pos}(\cdot)$ is a learnable position embedding function. Note that we omit the descriptions of layer normalization and residual connection in each sublayer. Please refer to (Vaswani et al., 2017; Lewis et al., 2020) for more details.

The above process iterates L_e times. Finally, we obtain the hidden states $\mathbf{H}_e^{(L_e)}$ of the input document, which will provide useful information for both table header generator and table body generator via attention mechanisms.

4.2 Table Header Generator

As mentioned previously, our model is designed to generate a table as a table header and a set of table body rows. To this end, we follow previous studies (Zhang et al., 2018; Su et al., 2019) to decomposes the table generation into two steps. We first propose table header generator, which is the same as the BART decoder and generates the table header $\mathrm{Y}^0 = y_1^0, y_2^0, ..., y_{|\mathrm{Y}^0|}^0$ as a token sequence.

Given the encoder hidden states $\mathbf{H}_e^{(L_e)}$, our generator produces the table header in an auto-regressive manner:

$$\mathbf{A}_{d0}^{(l)} = \mathrm{MultiHead}(\mathbf{H}_{d0}^{(l)}, \mathbf{H}_{d0}^{(l)}, \mathbf{H}_{d0}^{(l)}), \quad (3)$$

$$\overline{\mathbf{A}}_{d0}^{(l)} = \mathrm{MultiHead}(\mathbf{A}_{d0}^{(l)}, \mathbf{H}_e^{(L_e)}, \mathbf{H}_e^{(L_e)}), \quad (4)$$

$$\mathbf{H}_{d0}^{(l+1)} = \mathrm{FFN}(\overline{\mathbf{A}}_{d0}^{(l)}), \quad (5)$$

where $\mathbf{H}_{d0}^{(l)} \in \mathbb{R}^{|\mathrm{Y}^0| \times d}$ is the hidden states at the l-th layer. In the first layer, we initialize the j-th decoder input as the sum of $\mathrm{Word}(y_{j-1}^0)$, $\mathrm{Pos}(y_{j-1}^0)$, $\mathrm{Row}(y_{j-1}^0)$ and $\mathrm{Col}(y_{j-1}^0)$, where $\mathrm{Word}(\cdot)$ and $\mathrm{Pos}(\cdot)$ share parameters with text encoder, row embeddings $\mathrm{Row}(\cdot)$ and column embeddings $\mathrm{Col}(\cdot)$ will be described in the next subsection.

Finally, after iterating the above process for L_d times, we obtain the last-layer hidden states $\mathbf{H}_{d0}^{(L_d)} = \{\mathrm{H}_{d0,j}^{(L_d)}\}_{1 \le j \le |\mathrm{Y}^0|}$, where $\mathrm{H}_{d0,j}^{(L_d)}$ is used to output the j-th target token:

$$y_j^0 = \mathrm{argmax}(\mathbf{W}_o \mathrm{H}_{d0,j}^{(L_d)}), \quad (6)$$

where $\mathbf{W}_o \in \mathbb{R}^{|\mathrm{V}| \times \mathrm{d}}$ is a learnable parameter matrix, and V is the target vocabulary.

4.3 Table Body Generator

To generate a set of table body rows in parallel, we propose a novel semi-autoregression table body generator. It is also stacked with L_d layers, each of which consists of self-attention, cross-attention and feed-forward network sublayers. Particularly, it shares parameters with table header generator, so that it can directly exploit the hidden states of table header generator via self-attention. This generator produces M table body rows $\{\mathrm{Y}^m\}_{1 \le m \le M}$ in parallel, under the semantic guidence of $\mathbf{H}_e^{(L_e)}$ and $\{\mathbf{H}_{d0}^{(l)}\}_{1 \le l \le L_d}$. Particularly, we use a special $\langle \varnothing \rangle$ token to represent "no corresponding row". Formally, the m-th table body row, $\mathrm{Y}^m = y_1^m, y_2^m, ..., y_{|\mathrm{Y}^m|}^m$ is generated in an auto-regressive way:

$$\overline{\mathbf{H}}_{dm}^{(l)} = [\mathbf{H}_{d0}^{(l)}; \mathbf{H}_{dm}^{(l)}], \quad (7)$$

$$\mathbf{A}_{dm}^{(l)} = \text{MultiHead}(\mathbf{H}_{dm}^{(l)}, \overline{\mathbf{H}}_{dm}^{(l)}, \overline{\mathbf{H}}_{dm}^{(l)}), \quad (8)$$

$$\overline{\mathbf{A}}_{dm}^{(l)} = \text{MultiHead}(\mathbf{A}_{dm}^{(l)}, \mathbf{H}_{e}^{(L_e)}, \mathbf{H}_{e}^{(L_e)}), \quad (9)$$

$$\mathbf{H}_{dm}^{(l+1)} = \text{FFN}(\overline{\mathbf{A}}_{dm}^{(l)}), \quad (10)$$

where $\mathbf{H}_{dm}^{(l)} \in \mathbb{R}^{|\mathrm{Y}^m| \times d}$ is the hidden states for generating Y^m. Note that our generator exploits both hidden states of table header and previous tokens in the same row to produce a table body row.

Taking the sum of word embeddings and positional embeddings as inputs, the vanilla Transformer decoder can only generate a sequence but not a set. To generate a set of table body rows in parallel, we introduce M additional learnable embeddings called *Row Embeddings* (See the green squares in Figure 2) into inputs, guiding table body generator to produce different rows. Here, M is a predefined parameter that is usually larger than the maximum number of rows in training data. Furthermore, to enhance the semantic consistency between cells in the same column, we add another learnable embeddings named *Column Embeddings* (See the blue squares in Figure 2) into inputs. Column embeddings are similar to positional embeddings but are defined at the cell level. By doing so, tokens of cells in the same column are equipped with identical column embeddings.

Formally, with row and column embeddings, the initial hidden state of our generator becomes

$$\begin{aligned} \mathrm{H}_{dm,k}^{(0)} &= \text{Word}(y_{k-1}^m) + \text{Pos}(y_{k-1}^m) \\ &+ \text{Row}(y_{k-1}^m) + \text{Col}(y_{k-1}^m), \end{aligned} \quad (11)$$

where y_{k-1}^m is the $(k-1)$-th output token at the m-th table body row, and $\text{Row}(\cdot)$ and $\text{Col}(\cdot)$ are row and column embedding functions, respectively.

Through L_d times of hidden state updates, we obtain the last-layer hidden states $\{\mathbf{H}_{dm}^{(L_d)}\}_{1 \le m \le M}$. Finally, based on $\mathrm{H}_{dm,k}^{(L_d)}$, we obtain the token with maximum probability as the output:

$$y_k^m = \text{argmax}(\mathbf{W}_o \mathrm{H}_{dm,k}^{(L_d)}). \quad (12)$$

In order to maintain an equal number of cells in every row, table body generator keeps generating a row until the number of $\langle s \rangle$ matches that in the header.

4.4 Training

Training Loss As mentioned above, we decompose the generation of a table into two steps. Correspondingly, we define the training loss as:

$$\mathcal{L} = \lambda \mathcal{L}_h + (1 - \lambda)\mathcal{L}_b, \quad (13)$$

where λ is a hyper-parameter to balance the effect of *the table header generation loss* $\mathcal{L}_h$ and *the table body generation loss* $\mathcal{L}_b$.

As for $\mathcal{L}_h$, we follow common practice to define $\mathcal{L}_h$ as a cross-entropy loss between the predictive distributions of the generated table header and the target one:

$$\mathcal{L}_h = -\sum_{j=1}^{|\mathrm{Y}^0|} \log \hat{p}_j^0(y_j^0), \quad (14)$$

where $\hat{p}_j^0(\cdot)$ is the predictive probability of the j-th token in the table header Y^0 using teacher forcing.

Target Assignments Based on the First Cells We also define $\mathcal{L}_b$ as a cross-entropy loss between the predictive distributions of the generated table body rows and targets. However, there is no correspondence between each generated table body row and target during training, and hence we can not directly calculate $\mathcal{L}_b$. To deal with this issue, we learn from the recent studies on set generation (Carion et al., 2020; Ye et al., 2021; Xie et al., 2022) and propose to efficiently determine target row assignments according to the matching results between the first cells of generated table body rows and those of targets. Here, we use the first cell to represent the whole row, because it is usually unique and often contains the important information of a table such as a name and a primary key.

Concretely, we first use our model to generate the first cells of all table body rows. During this process, we obtain generation probability distributions $\{\mathbf{P}^m\}_{1 \le m \le M}$, where $\mathbf{P}^m = \{p_k^m\}_{1 \le k \le |\mathbf{P}^m|}$ and p_k^m is the predictive distribution at the k-th timestep for the m-th table body row. Particularly, we pad the set of target table body rows to size M with $\langle \varnothing \rangle$. Afterwards, we determine the target assignments via the bipartite matching between the generated table body rows and targets:

$$\hat{f} = \underset{f \in \mathrm{F}(M)}{\text{argmin}} \sum_{m=1}^{M} \mathcal{C}(\mathrm{Y}^{f(m)}, \mathbf{P}^m), \quad (15)$$

where $\mathrm{F}(M)$ denotes the set of all $M!$ one-to-one mapping functions and $f(\cdot)$ is a function aligning

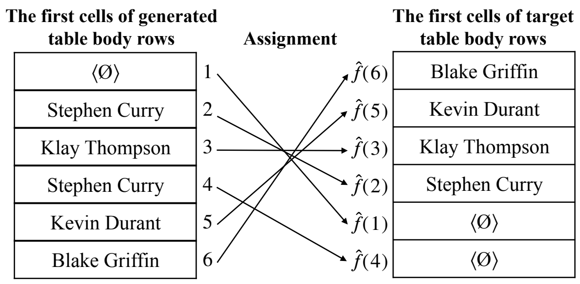


Figure 3: The bipartite matching between generated table body rows and targets. We use the first cell to represent each table body row, and determine target assignments according to the optimal bipartite matching between the generated first cells and those of targets. The function $\hat{f}(\cdot)$ maps the index of the input generated table body row to the corresponding target one.

the m-th generated table body row to the $f(m)$-th target one. The optimal matching can be efficiently determined with the Hungarian algorithm (Kuhn, 1955). More specifically, the matching cost $\mathcal{C}(\cdot)$ takes into account the token level probability, which is defined as follows:

$$\mathcal{C}(\mathrm{Y}^{f(m)}, \mathbf{P}^m) = -\sum_{k=1}^{N} \mathbb{1}_{\{\mathrm{Y} \neq \varnothing\}} p_k^m(y_k^{f(m)}), \tag{16}$$

where N is the length of the first cell in $\mathrm{Y}^{f(m)}$ and $p_k^m(y_k^{f(m)})$ is the predictive probability of the k-th target token $y_k^{f(m)}$ of the m-th table body row. We ignore the score from matching predictions with $\langle\varnothing\rangle$, so as to ensure that each generated row can be aligned with a non-empty target as possible.

For example, in Figure 3, our model generates the first cells of six table body rows, where the first one is $\langle\varnothing\rangle$ token and the others are player names. Then we assign each generated table body row to a target one according to the above-mentioned bipartite matching. In this example, we can find an optimal matching, with a mapping function $\hat{f}$ satisfying that $\hat{f}(1) = 5, \hat{f}(2) = 4, ..., \hat{f}(6) = 1$. Note that there are two "*Stephen Curry*" occurring in row 2 and row 4, but are aligned to different targets due to the one-to-one matching. In this way, we can guarantee that supervision signal for generating each table body row is unique, alleviating the generation of duplicate rows.

Finally, through the above target row assignments, we can calculate $\mathcal{L}_b$ as follows:

$$\mathcal{L}_b = -\sum_{m=1}^{M} \sum_{k=1}^{|\mathrm{Y}^{\hat{f}(m)}|} \log \hat{p}_k^m(y_k^{\hat{f}(m)}), \tag{17}$$

where $\hat{p}_k^m(\cdot)$ is the predictive distribution of the m-th generated table body row at the timestep k, and $y_k^{\hat{f}(m)}$ is the k-th token in the assigned target row.

Particularly, inspired by the recent studies (Carion et al., 2020; Ye et al., 2021; Xie et al., 2022), when $y_k^{\hat{f}(m)} = \langle\varnothing\rangle$, we multiply its token-level loss with a predefined factor to down-weight its effect, so as to reduce the negative effect of excessive $\langle\varnothing\rangle$ tokens.

5 Experiments

5.1 Setup

Datasets Following the previous work (Wu et al., 2022), we conduct experiment on four commonly-used datasets for table-to-text generation: Rotowire (Wiseman et al., 2017), E2E (Novikova et al., 2017), WikiTableText (Bao et al., 2018), and WikiBio (Lebret et al., 2016). As the main dataset of our experiments, Rotowire has two types of tables named Team and Player. In Rotowire, each instance has multiple columns while the other three datasets have only two columns. We use the processed datasets from (Wu et al., 2022). The dataset statistics are listed in Appendix A.

Implementation Details We initialize our model with the pre-trained BART-base (Lewis et al., 2020), which consists of 6 encoder layers and 6 decoder layers. The number of multi-head attention is 12, the dimension of embedding and hidden state is 768, and the dimension of feed-forward network is 3,072. We reuse the vocabulary from the pre-trained BART-base model, whose size is 51,200. We use the Adam (Kingma and Ba, 2015) optimization algorithm with a fixed maximum number of tokens as 4,096. For different datasets, we set different numbers of row embeddings according to the maximum row numbers in training sets. We train a separate model on each dataset and select the model with the lowest validation loss. Hyperparameter settings are shown in Appendix B.

Baselines We compare our model with the following baselines mentioned in (Wu et al., 2022):

- **Sent-level RE** This model uses an existing method of relation extraction (RE) (Zhong and Chen, 2021) to extract information based on predefined schemas. It takes the first column and data cells as entities and the types of table header cells as relations.
- **Doc-level RE** It applies the same RE method,

Subset	Model	The first column F1			Table header F1			Data cell F1			Error
		Exact	Chrf	BERT	Exact	Chrf	BERT	Exact	Chrf	BERT	rate
Team	Sent-level RE	85.28	87.12	93.65	85.54	87.99	87.53	77.17	79.10	87.48	0.00
	Doc-level RE	84.90	86.73	93.44	85.46	88.09	87.99	75.66	77.89	87.82	0.00
	Seq2Seq	94.71	94.93	97.35	**86.07**	89.18	88.90	82.97	84.43	90.62	0.49
	Seq2Seq-c	94.97	95.20	97.51	86.02	89.24	89.05	83.36	84.76	90.80	0.00
	Seq2Seq&set	**96.80**‡	**97.10**‡	**98.45**‡	86.00	**89.48**	**93.11**‡	**84.33**‡	**85.68**‡	**91.30**‡	0.00
Player	Sent-level RE	89.05	93.00	90.98	86.36	89.38	93.07	79.59	83.42	85.35	0.00
	Doc-level RE	89.26	93.28	91.19	87.35	90.22	97.30	80.76	84.64	86.50	0.00
	Seq2Seq	92.16	93.89	93.60	87.82	91.28	94.44	81.96	84.19	88.66	7.40
	Seq2Seq-c	92.31	94.00	93.71	87.78	91.26	94.41	82.53	84.74	88.97	0.00
	Seq2Seq&set	**92.83**†	**94.48**†	**96.43**‡	**88.02**	**91.60**†	**95.08**†	**83.51**‡	**85.75**‡	**90.93**‡	0.00

Table 2: Results of baselines and our Seq2Seq&set model on Rotowire. We show the F1 score based on exact match (Exact), chrf score (Chrf), and BERTScore (BERT). †/‡ indicates significant at p<0.05/0.01 over Seq2Seq-c with 1000 bootstrap tests (Efron and Tibshirani, 1994).

except that it predicts the relations between entities within multiple sentences.

- **NER** It is a BERT-based (Devlin et al., 2019) entity extraction method that considers data cells in each table as entities and its first column cells as entity types.
- **Seq2Seq** (Wu et al., 2022) It is a Transformer based seq2seq model that models the generation of a table as a sequence.
- **Seq2Seq-c** (Wu et al., 2022) This model is a Seq2Seq variant, where the cell number of each table body row is limited to the same as that of table header.

Evaluation We use the same evaluation script from (Wu et al., 2022). We adopt the F1 score as the evaluation measure, which is calculated in the following way: the precision and recall are first computed to get table-specific F1 scores, which are then averaged to obtain the final score. Here, precision is defined as the percentage of correctly predicted cells among the generated cells, and recall is defined as the percentage of correctly predicted cells among target cells. Particularly, the F1 score is calculated in three ways: *exact match* that matches two cells exactly, *chrf score* that calculates character-level n-gram similarity, and *BERTscore* that calculates the similarity between BERT embeddings of two cells.

For Rotowire, we report the F1 scores of the first column, table header and data cells, which refer to *row header F1*, *column header F1* and *non-header cells F1* in (Wu et al., 2022). For the other three datasets, there are only fixed two columns, so the F1 score of table header is not calculated. For data cells, we use not only the content but also the table header/the first column cells to ensure that the cell is on the right column/row. Note that these metrics are insensitive to the orders of rows and columns. Besides, we calculate the error rate to represent the percentage of erroneous format tables.

Model	# Sentences per second (speedup)		
	Team	Player	E2E
Seq2Seq	1.24 (1.00×)	0.32 (1.00×)	1.66 (1.00×)
Seq2Seq-c	1.22 (0.98×)	0.30 (0.94×)	1.62 (0.98×)
Seq2Seq&set	1.84 (1.48×)	1.09 (3.41×)	5.96 (3.59×)

Table 3: Inference efficiency comparison among different models. Here, we conduct experiments on an NVIDIA RTX 3090 GPU.

5.2 Main Results

Table 2 reports the results on the Team and Player subsets of Rotowire. We observe that our model consistently outperforms all baselines in terms of three kinds of F1 scores. Particularly, in terms of data cell F1, which is the most difficult of the three kinds of F1 scores, ours achieves significant improvements. Besides, note that both Seq2Seq-c and our model enforce the number of cells in each table body row to be the same as that of table header, so their error rates are 0. Table 4 shows the results on E2E, WikiTableText and WikiBio. Likewise, our model outperforms almost all baselines.

We also provide a case in Appendix C to visually show the effectiveness of our model.

5.3 Inference Efficiency

We compare the inference efficiency of different models. From Table 3, we observe that ours is significantly more efficient than baselines, due to its advantage in the parallel generation of table body rows.

Dataset	Model	The first column F1 Exact	Chrf	BERT	Data cell F1 Exact	Chrf	BERT	Error rate
E2E	NER	91.23	92.40	95.34	90.80	90.97	92.20	0.00
	Seq2Seq	99.62	99.69	99.88	97.87	97.99	98.56	0.00
	Seq2Seq-c	**99.63**	**99.69**	**99.88**	97.88	98.00	98.57	0.00
	Seq2Seq&set	99.62	**99.69**	99.83	**98.65**‡	**98.70**‡	**99.08**‡	0.00
WikiTableText	NER	59.72	70.98	94.36	52.23	59.62	73.40	0.00
	Seq2Seq	78.15	84.00	95.60	59.26	69.12	80.69	0.41
	Seq2Seq-c	78.16	83.96	95.68	59.14	68.95	80.74	0.00
	Seq2Seq&set	**78.67**‡	**84.21**‡	**95.88**	**59.94**‡	**69.59**‡	**81.67**‡	0.00
WikiBio	NER	63.99	71.19	81.03	56.51	62.52	61.95	0.00
	Seq2Seq	80.53	84.98	92.61	68.98	77.16	76.54	0.00
	Seq2Seq-c	80.52	84.96	92.60	69.02	77.16	76.56	0.00
	Seq2Seq&set	**81.03**†	**85.44**†	**93.02**†	**69.51**‡	**77.53**‡	**77.13**‡	0.00

Table 4: Results of baselines and our Seq2Seq&set model on E2E, WikiTableText and WikiBio.

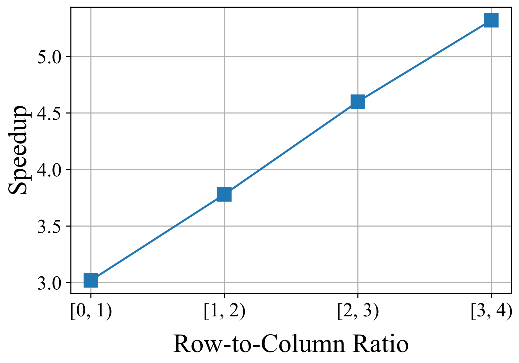


Figure 4: Speedup of Seq2Seq-c to Seq2Seq&set with different row-to-column ratios.

To investigate the effect of speedup on different types of tables, we carry out experiments on the Rotowire Player dataset. We define *row-to-column ratio* as the row number divided by the column number and measure the speedup with different row-to-column ratios. The results depicted in Figure 4 demonstrate that our model exhibits a linear improvement as the row-to-column ratio increases.

5.4 Error Propagation Analysis

As analyzed in Introduction, our model is able to better alleviate the error propagation issue than previous models. To verify this, we conduct experiments on the longest dataset Rotowire Player, and then compare the performance of our model and Seq2Seq-c. From Figure 5, we observe that ours always outperforms the baseline model. Particularly, with the number of table tokens increasing, our model exhibits more significant performance advantages over Seq2Seq-c.

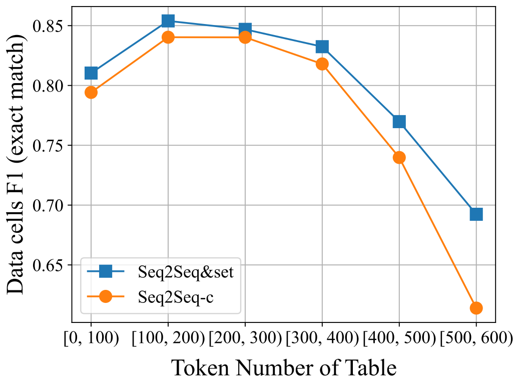


Figure 5: Performance comparison between Seq2Seq-c and Seq2Seq&set on the tables with different numbers of tokens.

Subset	Model	The first column F1	Table header F1	Data cell F1
Team	Seq2Seq&set	96.80	86.00	84.33
	w/o row embed.	66.91	85.73	57.25
	w/o col. embed.	96.75	86.04	83.32
	w/o tgt. assign.	92.87	85.87	79.43
	w/o header gen.	95.34	85.23	59.02
Player	Seq2Seq&set	92.83	88.02	83.51
	w/o row embed.	31.13	87.83	26.20
	w/o col. embed.	90.06	87.47	80.18
	w/o tgt. assign.	67.71	87.99	60.22
	w/o header gen.	92.45	86.83	58.39

Table 5: Results of ablation study. Here, the F1 scores are calculated in the way of exact match.

5.5 Ablation Study

We conduct an ablation study on Rotowire to verify the effectiveness of different components of our model. The results are shown in Table 5, involving four variants:

w/o Row Embeddings. We remove row embeddings in this variant. From lines 3 and 8, we observe that this variant is completely collapsed. This is reasonable that without row embeddings, the row-specific inputs of table body generator become exactly the same, resulting in the generator failing to generate distinct rows.

w/o Column Embeddings. We remove column embeddings in this variant. From lines 4 and 9, we observe that the data cell F1 decreases a lot. Thus, we also confirm that column embeddings are indeed useful in enhancing the semantic consistency between cells in the same column. The other two scores changed very little, which we believe is due to the fact that table headers and the first columns have no need to refer to the other rows.

w/o Target Assignments. In this variant, we discard the target assignments based on the first cells during training, which makes our model learn to generate table body rows in the original order of targets. As shown in lines 5 and 10, our model exhibits a significant performance drop.

w/o Table Header Generator. In this variant, we simultaneously generate table header and table body rows in parallel. Consequently, the variant can not leverage the information of generated table header during the process of generating table body rows, and thus exhibits worse performance than the original model. This result proves that it is reasonable to distinguish the generation of table header and table body rows.

6 Conclusion

In this paper, we propose a Seq2Seq&set model for text-to-table generation, which first outputs a table header, and then table body rows. Most importantly, unlike the previous study (Wu et al., 2022), we model the generation of table body rows as a set generation task, which is able to alleviate not only the wrong bias caused by the predefined order during training, but also the problem of error propagation during inference. Experimental results show that our model gains significant performance improvements over the existing SOTA.

Limitations

Our model is currently suitable for generating ordinary tables with attribute names and records, but it may struggle with more complex table formats that involve merged cells. To improve the flexibility of our model, we plan to investigate more versatile forms of table representation.

Another limitation of our model is that our model training involves longer training time, compared with seq2seq baselines. This may be due to the inherent instability of target assignments. In the future, we will explore refining the model training by reducing the target assignment instability.

The existing datasets for this task is relatively simple, and in the future we will conduct experiments on more complex datasets that require reasoning, such as WebNLG (Gardent et al., 2017).

Ethics Statement

This paper proposes a Seq2Seq&set model for text-to-table generation. We take ethical considerations seriously and ensure that the research and methods used in this study are conducted in an ethical and responsible manner. The datasets used in this paper are publicly available and have been widely adopted by researchers for testing the performance of text-to-table generation. This study does not involve any data collection or release, and thus there exist no privacy issues. We also take steps to ensure that the findings and conclusions of this study are reported accurately and objectively.

Acknowledgement

The project was supported by National Natural Science Foundation of China (No. 62276219), Natural Science Foundation of Fujian Province of China (No. 2020J06001), Youth Innovation Fund of Xiamen (No. 3502Z20206059). We also thank the reviewers for their insightful comments.

References

Junwei Bao, Duyu Tang, Nan Duan, Zhao Yan, Yuanhua Lv, Ming Zhou, and Tiejun Zhao. 2018. Table-to-text: Describing table region with natural language. In *Proc. of AAAI*.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *Proc. of ECCV*.

Lingzhen Chen and Alessandro Moschitti. 2018. Learning to progressively recognize new named entities with sequence to sequence models. In *Proc. of COLING*.

Wenhu Chen, Jianshu Chen, Yu Su, Zhiyu Chen, and William Yang Wang. 2020. Logical natural language generation from open-domain tables. In *Proc. of ACL*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL-HLT*.

Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. CRC press.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proc. of ICNLG*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICCV*.

Harold W Kuhn. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proc. of EMNLP*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proc. of ACL*.

Sha Li, Heng Ji, and Jiawei Han. 2021. Document-level event argument extraction by conditional generation. In *Proc. of NAACL-HLT*.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022a. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proc. of ACL*.

Yaojie Lu, Hongyu Lin, Jin Xu, Xianpei Han, Jialong Tang, Annan Li, Le Sun, Meng Liao, and Shaoyi Chen. 2021. Text2Event: Controllable sequence-to-structure generation for end-to-end event extraction. In *Proc. of ACL*.

Yaojie Lu, Qing Liu, Dai Dai, Xinyan Xiao, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. 2022b. Unified structure generation for universal information extraction. In *Proc. of ACL*.

Tapas Nayak and Hwee Tou Ng. 2020. Effective modeling of encoder-decoder architecture for joint entity and relation extraction. In *Proc. of AAAI*.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Proc. of SIGDIAL*.

Giovanni Paolini, Ben Athiwaratkun, Jason Krone, Jie Ma, Alessandro Achille, RISHITA ANUBHAI, Cicero Nogueira dos Santos, Bing Xiang, and Stefano Soatto. 2021. Structured prediction as translation between augmented natural languages. In *Proc. of ICLR*.

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 *Proc. of EMNLP*.

Sunita Sarawagi et al. 2008. Information extraction. *Foundations and Trends® in Databases*, 1(3):261–377.

Jinsong Su, Xiangwen Zhang, Qian Lin, Yue Qin, Junfeng Yao, and Yang Liu. 2019. Exploiting reverse target-side contexts for neural machine translation via asynchronous bidirectional decoding. *Artificial Intelligence*, 277:103168.

Zeqi Tan, Yongliang Shen, Shuai Zhang, Weiming Lu, and Yueting Zhuang. 2021. A sequence-to-set network for nested named entity recognition. In *Proc. of IJCAI*.

Craig Thomson, Ehud Reiter, and Somayajulu Sripada. 2020. SportSett:basketball - a robust and maintainable data-set for natural language generation. In *Proc. of the Workshop on IntelLanG*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NeurIPS*.

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2020. Neural machine translation with byte-level subwords. In *Proc. of AAAI*.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proc. of EMNLP*.

Xueqing Wu, Jiacheng Zhang, and Hang Li. 2022. Text-to-table: A new way of information extraction. In *Proc. of ACL*.

Binbin Xie, Xiangpeng Wei, Baosong Yang, Huan Lin, Jun Xie, Xiaoli Wang, Min Zhang, and Jinsong Su. 2022. WR-One2Set: Towards well-calibrated keyphrase generation. In *Proc. of EMNLP*.

Hang Yan, Tao Gui, Junqi Dai, Qipeng Guo, Zheng Zhang, and Xipeng Qiu. 2021. A unified generative framework for various NER subtasks. In *Proc. of ACL*.

Jiacheng Ye, Tao Gui, Yichao Luo, Yige Xu, and Qi Zhang. 2021. One2Set: Generating diverse keyphrases as a set. In *Proc. of ACL*.

Xiangrong Zeng, Daojian Zeng, Shizhu He, Kang Liu, and Jun Zhao. 2018. Extracting relational facts by an end-to-end neural model with copy mechanism. In *Proc. of ACL*.

Xiangwen Zhang, Jinsong Su, Yue Qin, Yang Liu, Rongrong Ji, and Hongji Wang. 2018. Asynchronous bidirectional decoding for neural machine translation. In *Proc. of AAAI*.

Zexuan Zhong and Danqi Chen. 2021. A frustratingly easy approach for entity and relation extraction. In *Proc. of NAACL-HLT*.

A Dataset Statistics

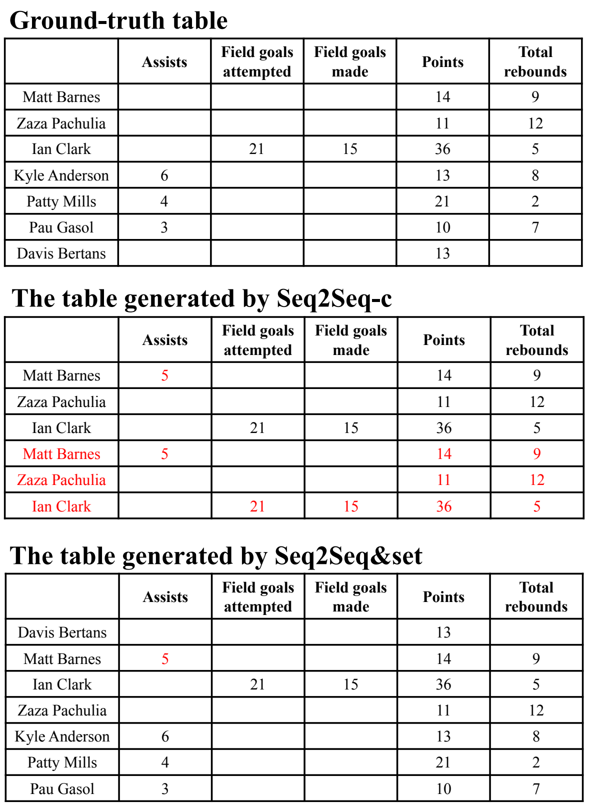
Table 6 shows the statistics of the datasets we used. We list the numbers of instances in training, validation, and test sets and the average number of BPE tokens per instance. We also give the average numbers of rows and columns per instance.

B Hyper-parameter Settings

Table 7 shows the hyper-parameter settings in our experiments. We set the hyper-parameters by referring to the existing work and choosing values that result in the best performance (measured in data cell F1) on the validation sets.

C Case Study

Figure 6 shows a case comparison between Seq2Seq-c and our Seq2Seq&set. Although the first three table body rows generated by Seq2Seq-c are almost correct, the others are duplicated, which also frequently occurs in other text generation tasks. In contrast, our model can handle this case correctly because ours generates table body rows in parallel and thus is not affected by other rows.



Ground-truth table

	Assists	Field goals attempted	Field goals made	Points	Total rebounds
Matt Barnes				14	9
Zaza Pachulia				11	12
Ian Clark		21	15	36	5
Kyle Anderson	6			13	8
Patty Mills	4			21	2
Pau Gasol	3			10	7
Davis Bertans				13	

The table generated by Seq2Seq-c

	Assists	Field goals attempted	Field goals made	Points	Total rebounds
Matt Barnes	5			14	9
Zaza Pachulia				11	12
Ian Clark		21	15	36	5
Matt Barnes	5			14	9
Zaza Pachulia				11	12
Ian Clark		21	15	36	5

The table generated by Seq2Seq&set

	Assists	Field goals attempted	Field goals made	Points	Total rebounds
Davis Bertans				13	
Matt Barnes	5			14	9
Ian Clark		21	15	36	5
Zaza Pachulia				11	12
Kyle Anderson	6			13	8
Patty Mills	4			21	2
Pau Gasol	3			10	7

Figure 6: A case study of Seq2Seq-c and our Seq2Seq&set model. Incorretly-generated texts are marked in red.

Dataset	Train	Valid	Test	Avg. # of tokens	Avg. # of rows	Avg. # of columns
Rotowire-Team	3.4k	727	728	351.05	2.71	4.84
Rotowire-Player	3.4k	727	728	351.05	7.26	8.75
E2E	42.1k	4.7k	4.7k	24.90	4.58	2.00
WikiTableText	10.0k	1.3k	2.0k	19.59	4.26	2.00
WikiBio	582.7k	72.8k	72.7k	122.30	4.20	2.00

Table 6: Statistics of Rotowire, E2E, WikiTableText and WikiBio datasets, including the number of instances in training, validation and test sets, the average number of BPE tokens per instance, and the average number of rows and columns per instance.

Dataset	M	λ	$\langle\varnothing\rangle$ scale	batch size	lr	warmup ratio
Rotowire-Team	10	1	0.2	4,096	1e-04	0.01
Rotowire-Player	20	1	0.4	2,048	1e-04	0.01
E2E	10	-	0.2	4,096	1e-05	0.1
WikiTableText	10	-	0.4	4,096	1e-04	0.1
WikiBio	25	-	0.1	2,048	1e-04	0.1

Table 7: The hyper-parameter settings in our experiments.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
7

☑ A2. Did you discuss any potential risks of your work?
8

☑ A3. Do the abstract and introduction summarize the paper's main claims?
1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

5

☑ B1. Did you cite the creators of artifacts you used?
5

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
5

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
5

☑ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
8

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
5

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
10

C ☑ Did you run computational experiments?

5

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
5

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
5

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
5

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
5

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?
Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.