Towards Faithful Response Generation for Chinese Table Question Answering

Anonymous ACL submission

Abstract

The response generation for TableQA aims to automatically generate a response to end-users from a SQL query and its corresponding execution result (in the form of table). It is an essential and practical task. However, there has been little work on it in recent years. We consider this may be blamed on the lack of large-scale and high-quality datasets in this area. In this paper, we present ResponseNLG, a large-scale and high-quality Chinese dataset for TableQA response generation, to advance the field in both academic and industrial communities. Further, to bridge the structural gap between the input SQL and table and establish better semantic alignments, we propose a Heterogeneous Graph Transformation approach. In this way, we establish a joint encoding space for the two heterogeneous input data and convert this task to a Graph-to-Text problem. We further introduce the Node Segment Embedding to better preserve the original graph structure upon PLMs based models.

1 Introduction

Table Question Answering (TableQA) aims to answer a question over the given tables, and it has been widely applied in many real-life applications, e.g., chatbot and business intelligence (Stent et al., 1999; Litman and Stillman, 2004; Budzianowski et al., 2018). One common solution is converting it to a Text-to-SQL problem (Warren and Pereira, 1982; Zettlemoyer and Collins, 2005; Mrksic et al., 2015), which maps the natural language question to meaning representations in SQL. Once a natural language question has been mapped to a formal SQL query, the result can be retrieved from the table database based on it. In a real-world setting, the consequent problem is how to convert the execution result, which usually can be organized as a table, to a natural language text to the asker, i.e., the response generation for TableQA (Yu et al., 2019a).

Figure 1: An example for Text-to-SQL and TableQA Response Generation. The red dotted lines denote the input data (a SQL query and its execution result) for TableQA Response Generation.

The response generation for TableQA takes a SQL query and its corresponding execution result (in the form of a table) as input and aims to generate a natural language description as the response (as shown in Figure 1). Intuitively, it plays a vital and indispensable role in constructing a real-life TableQA application and building a human-like dialog system. Meanwhile, this task is challenging. The first challenge is that the model needs to understand the two heterogeneous input data: SQL and table. Moreover, both input data are structured and have less semantic information than natural language sentences, which also exists in Table-to-Text generation (Lebret et al., 2016; Wiseman et al., 2017) but is more challenging. Additionally, the generated response must be absolutely faithful to the input data, which means the response should contain all the content in the input table while being logically consistent with the SQL.

To our knowledge, template-based models are widely applied in dialogue response generation modules (Jordan et al., 2006; Ultes et al., 2017).
The experts, who have abundant linguistic and domain knowledge, write different kinds of templates with slots which are then filled with the execution results (Ritter et al., 2011; Kale and Rastogi, 2020). Obviously, to cover more data from different domains, this system needs numerous templates, which typically require a lot of human effort and costs. Meanwhile, it is not easy to guarantee the fluency of the generated results. Over the past several years, automatic neural network-based methods have achieved significant progress in the text generation domain (Liu et al., 2016; Lubis et al., 2018). However, we notice that there is little work on response generation for TableQA. We consider this may be blamed on the lack of large-scale and high-quality datasets in this field. CoSQL (Yu et al., 2019a) is the only dataset for this task with 7,845 generation examples, and it is in English. It is a dataset with SQL-grounded dialogue state tracking as the core, and the generation annotations are very rough.

In this paper, we propose ResponseNLG, a large-scale and high-quality Chinese dataset for TableQA Response Generation. We introduce a dataset construction process where annotators only need to directly revise the provided template response, and yield 29,366 response generation examples. It is an order of magnitude larger than CoSQL. A strict screening procedure is implemented to ensure data quality. ResponseNLG has a wider distribution than CoSQL, which is more in line with real TableQA scenarios. Meanwhile, to bridge the structural gap between the input SQL and table and establish better semantic alignments, we propose a Heterogeneous Graph Transformation approach (HGT). HGT first converts the two sources to two undirected graphs and then builds the connection between the nodes in different graphs to obtain a heterogeneous joint graph. In this way, we convert this task to a Graph-to-Text problem. Previous Graph-to-Text methods (Ribeiro et al., 2020, 2021) transform the input graph into a new token graph to introduce pretrained language models (PLMs). We consider that this transformation breaks the input graph structure and may bring in extra noises into graph encoding. To preserve original structure information, we introduce the Node Segment Embedding, which assigns the same symbol to the nodes in the token graph which belong to the same node in the original heterogeneous graph. Our contributions include the following three aspects:

- We present a large-scale and high-quality Chinese dataset for TableQA response generation, ResponseNLG, with a series of strong baselines and metrics. To the best of our knowledge, it is also the first Chinese dataset for this task.
- We propose a Heterogeneous Graph Transformation method to bridge the structural gap between the SQL and table. We also introduce Node Segment Embedding to better preserve the original graph structure upon PLMs based models.
- Experiments and analysis on ResponseNLG attest to both the high quality and challenges of the dataset. The results also demonstrate the effectiveness of our proposed method. We will make our data and code publicly available upon the acceptance of this paper.

2 Related Works

2.1 Table Question Answering

A TableQA system comprises a table semantic parsing (Text-to-SQL) component and a response generation component (Yu et al., 2019a). The semantic parsing component converts NL question into SQL query (Text-to-SQL) (Finegan-Dollak et al., 2018; Guo et al., 2019; Wang et al., 2020a; Hui et al., 2021) and the response generation component generate NL response given the SQL query and SQL execution table. Notice that the SQL query can represent the context state in multi-turn TableQA scenarios (Yu et al., 2019a,b). Several datasets have been proposed to apply semantic parsing on tables, including WikiTableQuestions (Pasupat and Liang, 2015), SequentialQA (Iyyer et al., 2017), WikiSQL (Zhong et al., 2017), Spider (Yu et al., 2018), SparC (Yu et al., 2019b) and CHASE (Guo et al., 2021). But these works only focus on the semantic parsing task and return the SQL execution result as simple short form answer. FeTaQA (Nan et al., 2021) yields a more challenging TableQA setting because it requires generating free-form text answers. HybridQA (Chen et al., 2020b) and OTT-QA (Chen et al., 2020a) build question answering tasks with context of both structured tables and unstructured text.

2.2 Data-to-Text Generation

Data-to-Text aims to generate a natural language description from structural or semi-structural data.
(Liang et al., 2009; Banik et al., 2012; Gardent et al., 2017; Parikh et al., 2020). It helps people get the key points of the input data and makes the stored information accessible to a broader audience of end-users. In the academic community, Data-to-Text is usually divided into Graph-to-Text (Song et al., 2018; Wang et al., 2020c) and Table-to-Text (Lebret et al., 2016; Wiseman et al., 2017), according to whether input data is a graph (e.g., Knowledge or Abstract Meaning Representation Graph) or table (Ribeiro et al., 2019; Agarwal et al., 2021). To better model the structure of a graph, early works (Song et al., 2018; Koncel-Kedziorski et al., 2019; Damonte and Cohen, 2019) introduce Graph Neural Networks (GNNs) as the structure encoder, which only considers the relations between neighbor nodes. Unlike the local encoding strategies, Zhu et al.; Cai and Lam propose the Graph Transformer that uses explicit relation encoding and allows direct communication between two distant nodes. In order to learn better contextualized node embeddings, Ribeiro et al. gather the above two encoding strategies, proposing novel neural models which encode an input graph combining both global and local node contexts. To better leverage the structure of tables, some studies (Bao et al., 2018; Nema et al., 2018; Jain et al., 2018; Liu et al., 2019; Li et al., 2021) propose to utilize the hierarchical encoder to model the table’s representation from the row and column levels.

The response generation for TableQA can also be regarded as a Data-to-Text task, and it is similar to Table-to-Text but more challenging because its input data contains not only a structural table but also a SQL, which are both essential for the generation. Moreover, it requires the model to generate an utterly faithful response to the input data, which means the response should contain all the content in the table and be logically consistent with the SQL.

3 Dataset Construction

3.1 Data Synthetic and Annotation

Different from previous works (Yu et al., 2019a), which usually rely on humans to create natural Language (NL) questions, SQL queries and corresponding response, we introduce a dataset construction process where annotators only need to directly revise the provided template response as illustrated in Figure 2. We first collect tables from the Internet and utilize production rules to generate SQL queries automatically. And then, we execute the SQL on the collected tables. After that, we generate a pseudo response based on predefined templates. Finally, pseudo responses are paraphrased to NL responses by humans. Additionally, to guarantee data quality, low-confidence instances are detected according to their overlapping and similarity metrics and are further checked by humans.

Table Collection We build a search engine based table collection pipeline to collect high quality tables. Firstly, 100,000 frequently used words are summarized from the CLUE (Xu et al., 2020) corpus. Then these words are queried in Google and filtered spreadsheet files are downloaded. Useful tables are extracted from these files through a parser, which could identify potential tables in a worksheet. Sensitive values in the tables, such as
Passwords, Identification IDs and Credit Card IDs are replaced with special tokens. We also build a table cleaning pipeline as shown in Appendix A.1 to guarantee table quality.

**Pseudo Data Generation** Data syntactic for semantic parsing has gained increasing attention in recent years (Zhong et al., 2017; Wang et al., 2020b, 2021). Differently, we apply syntactic method to build response generation dataset. We firstly utilize production rules from the SQL grammar to automatically generate SQL queries. The SQL query can be represented as an abstract syntax trees (ASTs) using the rules such as SQLs = SQL, SQL = Select Where, Select = SELECT A, Where = WHERE Conditions.... all of which are production rules of the SQL grammar. Please refer to Appendix A.2 for more details. By exploiting every rule of the grammar, we can generate SQL queries covering patterns of different complexity along with corresponding tables. SQL queries which cannot execute or have not execution results are filtered. We then build two template-based generation pipelines. The one is to convert the syntactic SQL query into pseudo NL question. The other is to generate template NL response based on SQL query and the SQL execution result table.

**Data Annotation and Review** We employ 20 well-educated crowd workers to paraphrase the template questions and template response into natural language, and filter incomprehensible ones which are semantically unclear. To guarantee data quality, another 4 workers are asked to review the annotated data. Data with poor annotation quality will be required to be relabeled. We also automatically detect low-quality data. If the response does not contain important information about SQL and Table, we will filter it out.

**3.2 Dataset Statistics**

Our final ResponseNLG dataset contains 29,358 examples, with a average length of 46.7. Each example contains a {NL question, SQL query, SQL execution table, NL response} pair. We split the training/development/test set by 23,488/2,935/2,935 randomly.

**Topics** We build a topic categorization model (Asthana and Halfaker, 2018) for tables in ResponseNLG to investigate the topics distribution. Figure 3 presents an aggregated topic analysis of our dataset. We find that the Media, Insurance and Bank topics together comprise 61% of our dataset, but the other 39% is composed of broader topics such as Public Service, Technology, Finance. Our dataset is limited to topics that are present in CLUE.

**Data Complexity** We evaluate the data complexity by the row number and column number of the input tables. Figure 4 shows the training set distribution comparison between CoSQL and ResponseNLG. We can see that the ResponseNLG has a wider distribution than CoSQL, which is more in line with real TableQA scenarios. Please refer to Appendix A.3 for more details.

**4 Structure-Aware Approach**

Given an input SQL $s$ and a Table $t$, the model aims to generate a response $\hat{y}$. To bridge the gap between the two sources of information, we first propose a Heterogeneous Graph Transformation approach (HGT), which explicitly connects the input SQL and table in a heterogeneous graph structure. In this way, we can obtain a joint graph representation of the two sources and convert the response generation task to a Graph-to-Text problem. And then, we utilize a varietal transformer architecture (Ribeiro et al., 2020) that employs the original transformer encoder as the Global Node Encoder (G-NE) and introduces a GNN based layer into each transformer encoder layer as the Local Node Encoder (L-NE). G-NE allows explicit communication between two distant nodes, taking advantage of a large node context range. And L-NE has an advantage in modeling the graph topology. As shown in Figure 6 (b), this architecture cascaded performs global and local node aggregation, which gathers the benefits from both strategies. In the rest of this section, we will describe the proposed Heterogeneous Graph Transformation approach and the Local Node Encoder in detail.
4.1 Heterogeneous Graph Transformation

Given a SQL $s$ and its execution result (in the form of a table) $t$ as input (shown in Figure 1), the Heterogeneous Graph Transformation approach takes two steps to transform the input two sources of data into a heterogeneous graph (shown in Figure 6a). First, it converts the SQL and table into two undirected graphs: SQL graph $G_s$ and table graph $G_t$. In particular, for a SQL, we follow the previous method (Xu et al., 2018) and convert it to a tree. We refer the readers to the paper for more details.

For a table, we treat each column name and table cell as a node and divide the nodes in the table into two categories: table header node and table cell node. And then, we connect each header node with the cell node in the same column. We also build the connections between the cell nodes in the same row. Second, we add connections between the nodes that indicate the same column in $G_s$ and $G_t$ to build the unified heterogeneous graph. We also add a self-loop connection for each node. The transformed heterogeneous graph is formulated as $G_h = (V_h, E_h)$, where $V$ represents the nodes set and $E_h = \{(n, v)|n, v \in V\}$. Figure 6a shows an example of the transformed heterogeneous graph.

We expect that developing generation model should benefit from the recent advance on pre-trained language models (PLMs) (Peters et al., 2018; Devlin et al., 2019; Radford et al., 2019). We represent each $G_h$ using subword tokens, and convert it into a new token graph $G = (V, E)$. Specifically, each token of a node in $V_h$ becomes a node $\tilde{v}$ in $V$. For each edge $(n, v) \in E_h$, we connect each token between $n$ and $v$ to obtain the new edges set $\tilde{E}$ (as shown in Figure 5). However, we notice that the new token graph $\tilde{G}$ breaks the structure of the original graph $G_h$ and may make the encoder pay too much attention to the feature of nodes at the token level instead of the original node level. This may bring extra noises into graph encoding. To preserve the original structural information, we introduce the Node Segment Embedding (NSE), which assigns the same symbol to the nodes in the token graph $\tilde{G}$ which belong to the same node in the original heterogeneous graph $G_h$.

4.2 Local Node Encoder

Given $\{h_v|v \in V\}$ as the outputs of the Global Node Encoder at the $L$-th encoder layer, we next describe how the Local Node Encoder works. As shown in Figure 6b, the Local Node Encoder consists of two main modules: Node Embedding Layer and Graph Attention Network Layer. The former enriches the features of the nodes, and the latter explicitly models the graph structure. For Node Embedding Layer, in addition to the above Node Segment Embedding, we also introduce Node Type Embedding (NTE) to preserve the graph heterogeneity. Formally, given $h_v$, we obtain the feature-enhanced node representation by:

$$h_v^e = LayerNorm(h_v) + e_s^v + e_t^v \quad (1)$$

where $LayerNorm$ represents layer normalization (Ba et al., 2016). $e_s^v$, $e_t^v$ denote the node segment embedding and node type embedding for node $v$ respectively.

After the Node Embedding Layer, we utilize Graph Neural Networks (GNNs) to model the graph structure explicitly. For simplicity, we employ one Graph Attention Network Layer (GAT). Formally, it aggregates the representations of node $v$ in a multi-head self-attention layer (Vaswani, 2017).
et al., 2017) as follows:

\[
\begin{align*}
    h_{v,n}^h &= \frac{h_v^0 W_v^h (h_n^0 W_n^h)}{\sqrt{d/H}} \\
    a_{v,n}^h &= \sum_{\hat{n} \in \mathcal{N}(v)} e^{h_{\hat{n},n}} \\
    z^h &= \sum_{\hat{n} \in \mathcal{N}(v)} a_{v,n}^h (h_n^0 W_v^0) \\
    h^r &= \text{Concat}(z^1, \ldots, z^H)
\end{align*}
\]

where \(1 \leq h \leq H\), and \(W_v^0, W_n^0, W_v^h, W_n^h \in \mathbb{R}^{d \times (d/H)}\). \(\mathcal{N}(v)\) denotes the immediate neighborhood of node \(v\) in graph \(G\). We also tried the RGAT (Shaw et al., 2018). It performed comparable with GAT but introduced more parameters.

### 4.3 Training Objective

The transformer parameters are initialized with the pretrained T5 (Raffel et al., 2020), and the others are randomly initialized. Given each gold instance \((s, t, y)\), we fine-tune the model to optimize the following cross-entropy objective:

\[
\mathcal{L} = -\sum_{i=1}^{|y|} p(y_i | y_{1:i-1}; s, t)
\]

### 5 Experiment

#### 5.1 Experiment Settings

**Baselines** We conduct experiments on ResponseNLG and compare our method with several baselines, including:

- **Pointer-Generator** is an RNN-based Seq2Seq model with attention and copy mechanism. We concatenate the SQL and linearized table as input.

- **Finetune** denotes a Transformer encoder-decoder method which is initialized by T5. It takes the same strategy with Pointer-Generator to preprocess the input SQL and table. Moreover, we replace our local graph encoder with an FNN layer. And we change the hidden dimension of FNN and make its parameters equal with the local graph encoder to make a fair comparison. We denote this method as Finetune-FNN.

- **Finetune-Graph** is also a T5 initialized method. Different from Finetune, it uses the same graph linearization as input with our method. Additionally, we add FNN to make a fair comparison, which is denoted as Finetune-Graph-FNN.

**Evaluation Metrics** We evaluate our models by applying both automatic and human evaluations. For automatic evaluation, we first employ two widely used metrics: BLEU (Papineni et al., 2002) and CHRF++ (Popović, 2015). We also report the results of BLEU-2 and BLUE-4. All above scores are calculated by SacreBLEU (Post, 2018). Then we employ PARENT (Dhingra et al., 2019) to evaluate the faithfulness for the generated text. PARENT is a metric proposed specifically for data-to-text evaluation that takes the table into account. We modify it to make it suitable for our dataset, described in Appendix A.4. We conduct experiments over 4 different seeds and report the average scores on them. Please refer to Section 5.4 for human evaluation details.

**Implement Details** Our implementation is based on Hugging Face Transformer models (Wolf et al., 2020). We utilize T5base for all experiments.

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
<th>BLEU-2</th>
<th>BLEU-4</th>
<th>CHRF++</th>
<th>PARENT-P</th>
<th>PARENT-R</th>
<th>PARENT-R</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResponseNLG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Development</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pointer-Generator</td>
<td>50.30</td>
<td>54.40</td>
<td>37.30</td>
<td>59.46</td>
<td>53.22</td>
<td>80.95</td>
<td>62.91</td>
</tr>
<tr>
<td>Finetune</td>
<td>53.62</td>
<td>58.88</td>
<td>39.60</td>
<td>62.67</td>
<td>56.58</td>
<td>84.35</td>
<td>66.63</td>
</tr>
<tr>
<td>Finetune-FNN</td>
<td>54.32</td>
<td>59.53</td>
<td>40.25</td>
<td>63.06</td>
<td>56.67</td>
<td>84.49</td>
<td>66.73</td>
</tr>
<tr>
<td>Finetune-Graph</td>
<td>52.42</td>
<td>57.85</td>
<td>38.30</td>
<td>60.78</td>
<td>56.57</td>
<td>83.76</td>
<td>66.42</td>
</tr>
<tr>
<td>Finetune-Graph-FNN</td>
<td>53.03</td>
<td>58.45</td>
<td>38.63</td>
<td>60.97</td>
<td>56.79</td>
<td>83.98</td>
<td>66.64</td>
</tr>
<tr>
<td>Ours</td>
<td>55.88*</td>
<td>60.88*</td>
<td>42.15*</td>
<td>63.94*</td>
<td>56.75</td>
<td>84.81</td>
<td>66.91</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pointer-Generator</td>
<td>50.12</td>
<td>55.35</td>
<td>37.15</td>
<td>58.97</td>
<td>54.13</td>
<td>80.36</td>
<td>63.22</td>
</tr>
<tr>
<td>Finetune</td>
<td>53.78</td>
<td>58.98</td>
<td>39.78</td>
<td>62.81</td>
<td>56.63</td>
<td>84.25</td>
<td>66.65</td>
</tr>
<tr>
<td>Finetune-FNN</td>
<td>54.18</td>
<td>59.45</td>
<td>40.08</td>
<td>62.95</td>
<td>56.82</td>
<td>84.41</td>
<td>66.85</td>
</tr>
<tr>
<td>Finetune-Graph</td>
<td>52.77</td>
<td>58.18</td>
<td>38.43</td>
<td>61.07</td>
<td>56.52</td>
<td>83.63</td>
<td>66.37</td>
</tr>
<tr>
<td>Finetune-Graph-FNN</td>
<td>53.34</td>
<td>58.75</td>
<td>39.00</td>
<td>61.41</td>
<td>56.63</td>
<td>84.08</td>
<td>66.60</td>
</tr>
<tr>
<td>Ours</td>
<td>55.80*</td>
<td>60.78*</td>
<td>42.08*</td>
<td>64.02*</td>
<td>56.58</td>
<td>84.63</td>
<td>66.73</td>
</tr>
</tbody>
</table>

Table 1: Main results of models on ResponseNLG development set. * denotes the value is significantly different from other models at a \(p < 0.05\) level, according to an independent sample t-test.
For T5-based methods, we use AdamW optimizer (Loshchilov and Hutter, 2018) and employ a linearly decreasing learning rate schedule without warm-up. Moreover, the learning rate is fixed as $3e^{-5}$, and batch size is set as 4 for all experiments. We train the parameters from T5 and the added parameters together. During decoding, we employ beam search with a beam size 5. All experiments are implemented with Pytorch and trained on Nvidia Telsa V100 32GP GPUs.

### 5.2 Main Result

The results on ResponseNLG development and test sets are summarized in Table 1. First, we observe that after adding new parameters, Finetune-FNN and Finetune-Graph-FNN achieve better performance than their baselines. And then, we notice that Finetune-FNN performs better than Finetune-Graph-FNN, though their parameters are equal. We consider the reason is that the input of the former is more similar to natural language. This indicates that the representation of input data affects the model performance. Our method significantly outperforms Finetune-Graph-FNN on BLEU (+2.85) and CHRF++ (+2.97) and also obtains a higher PARENT score. It demonstrates that the improvement of our method not only comes from more parameters. Our approach also performs better than Finetune-FNN on BLEU (+1.56) and CHRF++ (+0.8) and achieves competitive results on PARENT. The results on the test set follow a pattern similar to the development set and our method achieves the start-of-the-art results on BLEU and CHRF++. It demonstrates the effectiveness of our proposed method.

### 5.3 Analysis and Discussion

#### Ablation Study

To examine the impact of each module in our method, we conduct the ablation study on ResponseNLG development set, and the results are shown in Table 2. Finetune-Graph-L-NE denotes the method that replaces each FNN module in Finetune-Graph-FNN with a GAT layer. As can be seen, the most improvement comes from the explicitly modeling of the graph structure. Moreover, both Node Type Embedding (NTE) and Node Segment Embedding (NSE) can improve the model’s performance. However, they reduce the model’s performance on PARENT. We think that it may be due to fluctuation of the PARENT metric.

#### Effects of input SQL and Table

In order to examine the effects of different input data, we conduct further experiments by removing the input SQL and Table. The results are summarized in Table 3. We first remove the SQL and only utilize the Table as input. As we can see, both Finetune-FNN and our methods perform poorly on all metrics. Then, we only use SQL as the model input. The performance degrades even more. The results demonstrate that both input SQL and table are essential for the response generation. It worth noting that Finetune-FNN and our method still obtain high PARENT scores after removing the Table input. It is unreasonable because each ground-truth response must contain all content in the input table (high coverage rate) to achieve a high faithfulness (refer to Section 5.4). Therefore, we think PARENT may not accurately measure the faithfulness of the text in ResponseNLG. We also notice that, after removing the input SQL, our method still performs better than Finetune-FNN. The result indicates that, in addition to using a hierarchical encoder, it may be a good choice to transform a table into a graph representation to model its structure in Table-to-Text. We leave this for future work.

#### Impact on the Table Complexity

In order to have a deeper understanding of the model’s performance, we further explore the mode performance under various numbers of rows and columns of the input table on the ResponseNLG development set. Figure 7 shows the BLEU comparison between our model and baselines. The BLEU scores of all the models decrease as the number of table rows or columns increases. Intuitively, the more rows or columns the table contains, the more complex the
We conduct human evaluation following Parikh et al. (2020). We compare our method with Pointer-Generator, Finetune-FNN and Ortacel. Specifically, we first randomly select 100 examples from the ResponseNLG test set and the corresponding outputs generated by each model. And then, four annotators are asked to evaluate the quality from the following four axes:

- **Fluency**: a sentence is fluent if it is grammatical and natural. And it is scored from 1 to 10, where 1 represents not Fluent, and 10 represents Mostly Fluent.
- **Faithfulness**: a sentence is considered faithful if it is logically consistent with the input SQL and all pieces of information are supported by the table. The score ranges from 1 to 10.
- **Coverage**: percentage of cells in the input table the candidate sentence covers. It is calculated by \( \frac{n_c}{n_t} \), where \( n_t \) denotes all cells in the input table, and \( n_c \) represents the number of cells covered by the sentence.
- **Repetition** number of cells the candidate sentence repeats. If a cell is repeated \( n \) times, it will be recorded \( n \) times.

We also introduce the reference as one candidate. And its results can be regarded as the upper bound (denoted as Oracle). For each sample, the annotators need to evaluate four sentences based on the input data. And they do not know which model generates these sentences. The final score for each criterion is the average from all annotators.

The results summarized in Table 4 show that the **Oracle** consistently achieves high performance than generation methods. It attests to the high quality of our human annotations. Our method outperforms baselines on almost all axes. It demonstrates the effectiveness of our proposed method. Although our model achieves a high coverage rate (93.15%), its Faithfulness score is relatively low (only 7.83), and there is a considerable gap compared with the Oracle. It indicates simply copying content from the input table can not guarantee the faithfulness of the generated response. It may be necessary for the model to understand the input SQL and table deeper, which is the biggest challenge in this dataset.

### 6 Conclusion

We present ResponseNLG, a large-scale and high-quality Chinese dataset for TableQA response generation, along with a series of baselines and metrics. We build a Heterogeneous Graph Transformation method to bridge the structural gap between the SQL and table. Meanwhile, to better use PLMs, we introduce the Node Segment Embedding to solve the problem that transforming the input graph to a new token graph breaks the original graph’s structure. Experiments on our ResponseNLG dataset show that our proposed model outperforms existing baseline models. We will make our data and code publicly available upon the acceptance of this paper.
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 Thirty-Second AAAI Conference on Artificial Intelligence.


the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), pages 622–627.


A Appendix

A.1 Table Data Cleaning
We build a rule-based table cleaning pipeline to guarantee table quality. We filter out noise tables via rules as follows. There are 24K tables in our dataset.

- **Blacklist Filtering** We first builds a blacklist including special chars, dirty words, emojis, and HTML words. And filter tables if the headers or the values include any word in the blacklist.

- **Header Type Filtering** We recognize all of the header types in each table including Text, Number, Time, and Bool. If the proportion of Text type is less than 30%, we filter out the table.

- **Complexity Filtering** We will filter out tables with less than 2 columns or rows.

- **Repetition Filtering** If a value repeats more than 50% in a table, we will filter out the table.

A.2 SQL Query Generation
We utilize production rules from the SQL grammar to automatically generate SQL queries inspired by Zhong et al. (2017); Wang et al. (2020b). As illustrated in Table 5, the SQL query can be represented as a tree using the rule sequence. All of which are production rules of the SQL grammar. By exploiting every rule of the grammar, we can generate SQL queries covering patterns of different complexity.

A.3 Example Data
The response of more than 2 rows of table in CoSQL will degenerate into a template response as shown in Figure 8a. Differently, we ask the annotator to write all the input information in the response. As shown in Figure 8b, the execution result table is fully described in ResponseNLG. Which is more in line with real TableQA scenarios.

A.4 PARENT Metric
PARENT (Dhingra et al., 2019) is a metric proposed specifically for Data-to-Text generation to evaluate the faithfulness of the generated texts. It takes the input graph or table into account. However, it cannot be directly applied to TableQA Response Generation because it does not consider the input SQL. To solve this problem, we convert each condition in the SQL into a <attribute, value> pair. Similarly, we also convert each cell in the input table into a <attribute, value> where attribute is the column name and value is the cell value. Figure 9 shows an example. However, according to our human evaluation, this metric is imperfect and does not consider the relation between attribute and value in each pair, especially the pair from the SQL. The modified code will be released with our dataset.

A.5 Case Study
Figure 10 shows an example case generated by FINE-TUNE and our final model. The result table has four rows and two columns, and the FINE-
<table>
<thead>
<tr>
<th>Company Name</th>
<th>Market Capitalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tencent</td>
<td>45,530</td>
</tr>
<tr>
<td>Alibaba</td>
<td>40,738</td>
</tr>
<tr>
<td>Meituan</td>
<td>14,589</td>
</tr>
<tr>
<td>Pinduoduo</td>
<td>14,216</td>
</tr>
</tbody>
</table>

SQL
```
SELECT Company NAME, Market Capitalization WHERE Market Capitalization > 10,000 AND Country = China
```

Figure 9: Example case of PARTENT metric.

Figure 10: Example case of different generation models

TUNE model only describe three rows of the results. Differently, our model can be completely faithful to a given SQL query and table, even for relatively large data.