PG-GSQL: Pointer-Generator Network with Guide Decoding for Cross-Domain Context-Dependent Text-to-SQL Generation

Huajie Wang[1*], **Mei Li**[1*], **Lei Chen**[1,2†]
[1]School of Computer Science and Technology, East China Normal University, Shanghai, China
[2]Shanghai Key Laboratory of Multidimensional Information Processing, Shanghai, China
{51184506041,51184506023}@stu.ecnu.edu.cn, lchen@cs.ecnu.edu.cn

Abstract

Text-to-SQL is a task of translating utterances to SQL queries, and most existing neural approaches of text-to-SQL focus on the cross-domain context-independent generation task. We pay close attention to the cross-domain context-dependent text-to-SQL generation task, which requires a model to depend on the interaction history and current utterance to generate SQL query. In this paper, we present an encoder-decoder model called PG-GSQL based on the interaction-level encoder and with two effective innovations in decoder to solve cross-domain context-dependent text-to-SQL task. 1) To effectively capture historical information of SQL query and reuse the previous SQL query tokens, we use a hybrid pointer-generator network as decoder to copy tokens from the previous SQL query via pointer, the generator part is utilized to generate new tokens. 2) We propose a guide component to limit the prediction space of vocabulary for avoiding table-column dependency and foreign key dependency errors during decoding phase. In addition, we design a column-table linking mechanism to improve the prediction accuracy of tables. On the challenging cross-domain context-dependent text-to-SQL benchmark SParC, PG-GSQL achieves 34.0% question matching accuracy and 19.0% interaction matching accuracy on the dev set. With BERT augmentation, PG-GSQL obtains 53.1% question matching accuracy and 34.7% interaction matching accuracy on the dev set, outperforms the previous state-of-the-art model by 5.9% question matching accuracy and 5.2% interaction matching accuracy. Our code is publicly available[1].

1 Introduction

Text-to-SQL is a sub-task of semantic parsing, which aims to convert utterances to SQL queries. A great deal of deep learning approaches (Xu et al., 2018; Hwang et al., 2019; Bogin et al., 2019; Guo et al., 2019; Wang et al., 2019) have been proposed to solve context-independent text-to-SQL tasks such as WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018c). Among them, SQLova (Hwang et al., 2019) achieves 84.2% and 83.6% logical form accuracy on WikiSQL dev and test sets, and RAT-SQL (Wang et al., 2019) achieves 69.7% and 65.6% exact matching accuracy on the Spider dev and test sets when accessing database content, respectively.

However, in real-world interaction scenarios, users often interact with the system through multi-turn dialogue. Then, the system needs to generate complete SQL query based on historical interaction information and current user utterance, as shown in Figure 1. Suhr et al. (2018) propose a seq2seq model with an interaction-level encoder to solve domain-specific context-dependent task called ATIS (Hemphill et al., 1990) which only contains the flight-booking domain. Unlike ATIS dataset, Yu et al. (2019b) present SParC, a new complex cross-domain and context-dependent text-to-SQL dataset based on Spider. Most of previous approaches in the field of text-to-SQL only focus on translating stand-alone utterances to SQL queries, and are not suitable for solving SParC which contains contextual dependencies.

[*]Equal contribution.
[†]Corresponding author.
[1]https://github.com/cfhaiteeh/PG-GSQL

Proceedings of the 28th International Conference on Computational Linguistics, pages 370–380
Barcelona, Spain (Online), December 8-13, 2020

Database: tvshow
Table: tv_channel, tv_series, cartoon

Utterance 1: Tell me the package option for the series named "Rock TV".
SQL query1: SELECT package_option FROM tv_channel WHERE series_name = "Rock TV"

Utterance 2: Tell me the language of this series.
SQL query2: SELECT language FROM tv_channel WHERE series_name = "Rock TV"

Utterance 3: List the language used least number of TV Channel. List language and number of TV Channel.
SQL query3: SELECT language , COUNT(*) FROM tv_channel GROUP BY language ORDER BY COUNT(*) ASC LIMIT 1

Figure 1: An example of interaction from SParC dataset (Yu et al., 2019b). The generation of each SQL query depends on the historical information and current utterance, except for the first SQL query.

CD-Seq2Seq (Yu et al., 2019b) is a baseline model of SParC and it do not consider the table-column dependency, which causes the errors in predicting columns when tables are mentioned in the question but columns show ambiguity. Recently, Zhang et al. (2019) propose EditSQL which contains an editing mechanism that treats the previous SQL query as a sequence and reuses the previous SQL query to achieve the new state-of-the-art performance on SParC. But EditSQL generates table and column together (e.g., tv_channel.series_name) in a token during decoding phase, which raises noise of table prediction. Moreover, we observe that foreign key dependency is significant in cross-domain text-to-SQL datasets (Yu et al., 2018c; Yu et al., 2019a; Yu et al., 2019b) which contain more than 95% SQL queries that depend on foreign keys to link tables in both train and dev sets. Last but not least, to our knowledge, there is no research on utilizing tables to guide the prediction of SQL queries in decoding phase. It is worth studying whether this method will improve the performance.

In this work, we propose a novel model called PG-GSQL to address the cross-domain context-dependent text-to-SQL task. Our encoder is based on interaction-level encoder, and we use pointer-generator network (PG) with guide (G) component as the decoder. The interaction-level encoder and pointer-generator network are employed to capture the historical information in encoding and decoding phases, respectively. The pointer part in pointer-generator network is used to copy a token from previous SQL query and the generator is used to generate a new token from vocabulary in each decoding step. The guide component is designed to avoid dependency errors which contain table-column dependency and foreign key dependency errors during token generation phase. In addition, we observe that linking table and its corresponding columns improves the prediction accuracy of tables, hence we design a novel column-table linking mechanism to concatenate table and its corresponding columns together to represent the table. Experimental results show that PG-GSQL achieves 34.0% question matching accuracy and 19.0% interaction matching accuracy on SParC dev set. When using BERT augmentation, PG-GSQL obtains 53.1% question matching accuracy and 34.7% interaction matching accuracy on SParC dev set, gains 5.9% question matching accuracy and 5.2% interaction matching accuracy improvements compared to the previous state-of-the-art model.

2 Related Work

The task of translating utterance to SQL query has a long history (Warren and Pereira, 1981; Androutsopoulos et al., 1995; Popescu et al., 2004; Giordani and Moschitti, 2012; Wang et al., 2017), and the earlier works focus on specific database and add extra manual intervention to generalize each database (Warren and Pereira, 1981; Li and Jagadish, 2014; Yaghmazadeh et al., 2017). Recently, with the advent of large-scale cross-domain text-to-SQL datasets (Zhong et al., 2017; Yu et al., 2018c; Yu et al., 2019a; Yu et al., 2019b), methods based on neural network are applied to the context-independent text-to-SQL tasks such as WikiSQL and Spider (Xu et al., 2018; Dong and Lapata, 2018; Yu et al., 2018a; Yu et al., 2018b; Hwang et al., 2019; Bogin et al., 2019; Guo et al., 2019; Wang et al., 2019).

However, the work most relevant to our research is the cross-domain context-dependent text-to-SQL

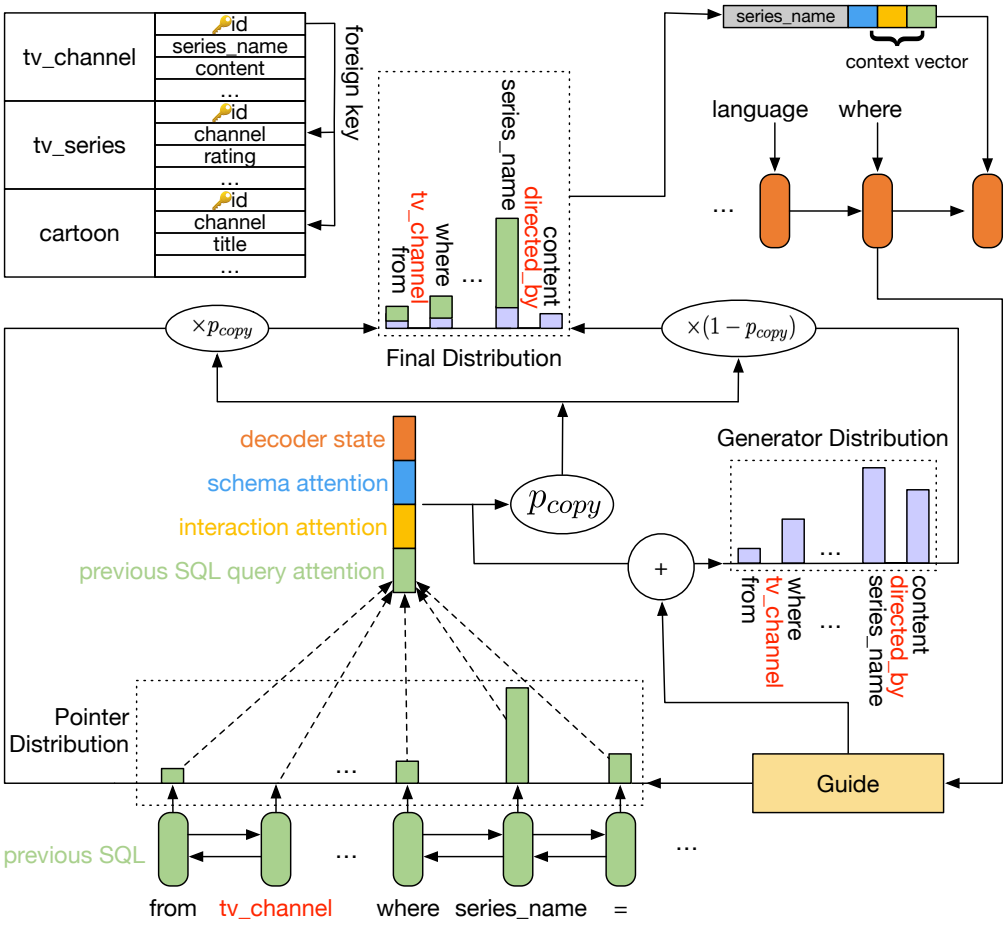


Figure 2: The decoder of our model, which is based on pointer-generator network with the guide component. The guide component prunes columns which do not belong to table `tv_channel` and tables when computing the output distribution of columns.

task. SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a) are the latest datasets of cross-domain context-dependent text-to-SQL, and CoSQL is more complex than SParC because CoSQL includes system responses. There is not enough research in cross-domain context-dependent text-to-SQL task, which requires a model to depend on the historical information and current utterance to generate SQL query from the unseen database. Suhr et al. (2018) propose a seq2seq model with an interaction-level encoder to solve ATIS (Hemphill et al., 1990), and this model is extended to SParC called CD-Seq2Seq (Yu et al., 2019b). CD-Seq2Seq uses position embeddings to get the position information for each utterance, and proposes a copy mechanism to copy segments from the previous SQL query. In addition, the interaction-level encoder uses a discourse state to maintain and update over entire interaction. Yu et al. (2019b) propose SyntaxSQL-con based on SyntaxSQL (Yu et al., 2018b), the difference between them is that SyntaxSQL-con uses two bi-directional LSTMs (Hochreiter and Schmidhuber, 1997) with different parameters to encode the previous utterance and current utterance. Both of them utilize the SQL syntax and generation history to generate SQL queries. Recently, Zhang et al. (2019) propose EditSQL to employ the interaction-level encoder with the utterance-table encoder as the encoder. Moreover, EditSQL proposes a table-aware decoder with an editing mechanism as the decoder. The editing mechanism is similar to our hybrid pointer-generator network, which is used to copy tokens from the previous SQL query or insert new tokens.

Our hybrid pointer-generator network is inspired by (See et al., 2017) which focuses on abstractive text summarization. The difference between our model and the previous model in pointer-generator network is that we copy words from previous SQL query rather than the source sentences.

3 PG-GSQL

In this section, we present the model PG-GSQL to tackle the cross-domain context-dependent text-to-SQL task. Similar to CD-Seq2Seq (Yu et al., 2019b), we employ an interaction-level encoder as PG-GSQL encoder. The decoder consists of two effective innovations. (1) We use the pointer-generator network as decoder to copy tokens from previous SQL query or generate new tokens from vocabulary. (2) We use guide component to limit the prediction space of vocabulary during decoding phase, which is able to avoid dependency errors to improve the performance. The architecture of decoder is illustrated in Figure 2. In addition, we use a column-table linking mechanism to concatenate table and its corresponding columns for each table to augment the representation of tables.

3.1 Schema Embedding

Let $s = \{t_1, \cdots, t_n\}$ denotes the set of tables in schema, n is the size of table set. Let $t_i = \{c_{i1}, \cdots, c_{im}\}$ denotes the column set, the elements in the set belong to table t_i and m is the size of the set. We use two different patterns to encode table and column, respectively. Figure 3 illustrates an example of our method. For column embeddings, we use a bi-directional LSTM (Hochreiter and Schmidhuber, 1997) to encode the concatenation of its type $column$ and the column name for each column (e.g., column series name), and use the final hidden state as column embedding. For table embeddings, we concatenate the column names in table t_i and table name with their types to represent table t_i (e.g., column id . column series name . table tv channel), which is able to augment the relation between column and table. Then, we use the same bi-directional LSTM to get the table t_i embedding. The schema embedding $\mathbf{h}^C$ consists of table embeddings and column embeddings.

3.2 Interaction-level Encoder

To obtain the relevance between utterance and schema, we use a simple string-matching algorithm to discern tables and columns which are mentioned in an utterance. We prioritize longer matching result and assign the type is `None` if there is no matching. If there are both column and table with same name in an utterance, we prioritize table. Let $x_i = [(x_{i,1}, \tau_{i,1}), \cdots, (x_{i,L}, \tau_{i,L})]$ denotes the matching result of an utterance in turn i and L is the length of x_i, the matching type of $x_{i,j}$ is $\tau_{i,j}$. First, we modify the original sequence as $[\tau_{i,1}, x_{i,1}$ ... $\tau_{i,L}, x_{i,L}]$ and denote $e_x^{i,j}$ as the j^{th} token embedding of the new sequence in turn i. Then we use a bi-directional LSTM to encode the new sequence and the forward LSTM is defined by:

$$\mathbf{h}_{i,j}^{\overrightarrow{E}} = \text{LSTM}^{\overrightarrow{E}}\left(\left[e_x^{i,j}; \mathbf{h}_{i-1}^{I}\right], \mathbf{h}_{i,j-1}^{\overrightarrow{E}}\right) \quad , \tag{1}$$

where $\mathbf{h}_{i,j}^{\overrightarrow{E}}$ is the j^{th} forward hidden state in turn i, and $\mathbf{h}_{i-1}^{I}$ is the turn $i-1$ discourse state which is used in interaction-level encoder to maintain and update over the entire interaction. The backward LSTM $\mathbf{h}_{i,j}^{\overleftarrow{E}}$ is modified analogously and the hidden state of $e_x^{i,j}$ is $\mathbf{h}_{i,j}^{E} = [\mathbf{h}_{i,j}^{\overrightarrow{E}}; \mathbf{h}_{i,j}^{\overleftarrow{E}}]$. We concatenate the hidden states of forward LSTM and backward LSTM at the last time as the input and update the discourse state

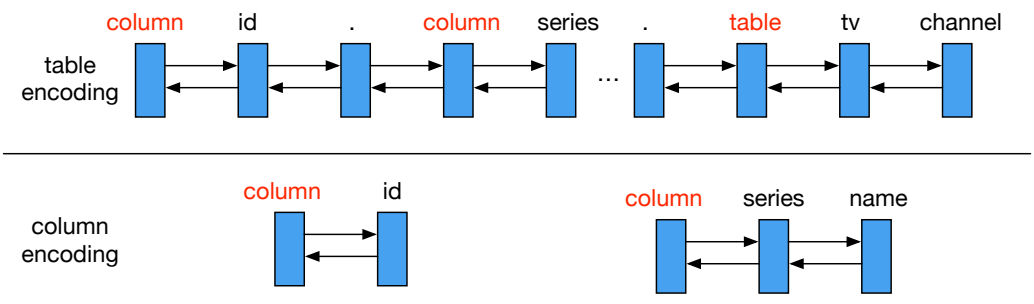


Figure 3: An example of table encoding and column encoding.

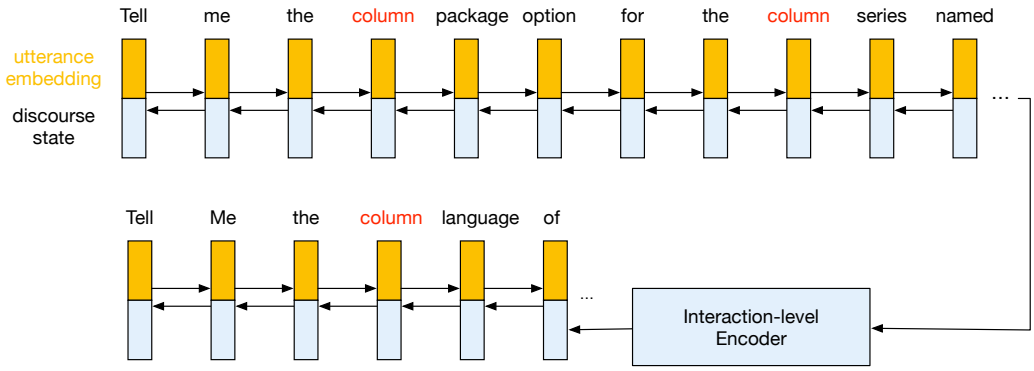


Figure 4: The interaction-level encoder architecture.

as $\mathbf{h}_i^I = \text{LSTM}^I\left([\overrightarrow{\mathbf{h}_{i,L}^E}; \overleftarrow{\mathbf{h}_{i,1}^E}], \mathbf{h}_{i-1}^I\right)$. The architecture of the interaction-level encoder is illustrated in Figure 4.

3.3 Base Decoder

Our base decoder is a LSTM decoder with attention (Bahdanau et al., 2015; Luong et al., 2015) to generate SQL queries. The decoder hidden state in step k is computed as:

$$\mathbf{h}_k^D = \text{LSTM}^D\left(\left[e_y^{k-1}; \mathbf{c}_{k-1}\right], \mathbf{h}_{k-1}^D\right) \quad , \tag{2}$$

where $\mathbf{h}_k^D$ is the decoder hidden state in step k, e_y^{k-1} is the $(k-1)^{th}$ output token embedding and $\mathbf{c}_{k-1}$ is the context vector in step $k-1$. The context vector $\mathbf{c}_k$ is the concatenate of interaction attention and schema attention vectors:

$$\mathbf{c}_k = [\mathbf{c}_k^{token}; \mathbf{c}_k^{schema}] \quad , \tag{3}$$

where $\mathbf{c}_k^{token}$ is the interaction attention vector and $\mathbf{c}_k^{schema}$ is the schema attention vector. We use the current utterance hidden state and the h previous utterance hidden states as the interaction encoder hidden state. In addition, we add relative position embeddings ϕ^I to each utterance hidden state in attention compution. The attention between the decoder hidden state and the interaction encoder hidden state is computed as:

$$\begin{aligned} s_k(t,j) &= (\mathbf{v}^{di})^T \tanh(\mathbf{W}_1^{di}\mathbf{h}_k^D + \mathbf{W}_2^{di}[\mathbf{h}_{t,j}^E; \phi^I(i-t)]) \\ \alpha_k^{token} &= \text{softmax}(s) \\ \mathbf{c}_k^{token} &= \sum_{t=i-h}^{i} \sum_{j=1}^{|x_t|} \alpha_k^{token}(t,j)[\mathbf{h}_{t,j}^E; \phi^I(i-t)] \quad , \end{aligned} \tag{4}$$

where α_k^{token} is the attention distribution and $|x_t|$ means the length of x_t. $\mathbf{v}^{di}$, $\mathbf{W}_1^{di}$ and $\mathbf{W}_2^{di}$ are learnable parameters. Furthermore, the attention between the decoder hidden state and the schema embedding is calculated as follows:

$$\begin{aligned} s_k(l) &= (\mathbf{v}^{dc})^T \tanh(\mathbf{W}_1^{dc}\mathbf{h}_k^D + \mathbf{W}_2^{dc}\mathbf{h}_l^C) \\ \alpha_k^{schema} &= \text{softmax}(s) \\ \mathbf{c}_k^{schema} &= \sum_{l=1}^{|schema|} \alpha_k^{schema}(l)\mathbf{h}_l^C \quad , \end{aligned} \tag{5}$$

$|schema|$ denotes the length of $\mathbf{h}^C$, $\mathbf{v}^{dc}$, $\mathbf{W}_1^{dc}$ and $\mathbf{W}_2^{dc}$ are learnable parameters. The probabilities of generating schema entries and SQL keywords are computed as:

$$\begin{aligned} \mathbf{o}_k &= [\mathbf{h}_k^D; \mathbf{c}_k]\mathbf{W}_o \\ S_{\text{SQL}} &= \mathbf{o}_k\mathbf{W}_{\text{SQL}} + \mathbf{b}_{\text{SQL}} \\ S_{\text{schema}} &= \mathbf{o}_k\mathbf{W}_{\text{schema}}\mathbf{h}^C \\ P_{vocab} &= \text{softmax}([S_{\text{SQL}}; S_{\text{schema}}]) \ , \end{aligned} \tag{6}$$

where S_{SQL} and S_{schema} are SQL keyword scores and schema entry scores, respectively. $\mathbf{W}_o$, $\mathbf{W}_{\text{SQL}}$, $\mathbf{b}_{\text{SQL}}$ and $\mathbf{W}_{\text{schema}}$ are learnable parameters.

3.4 Pointer-generator Network

We empirically verify that the current SQL query is similar to the previous SQL query and the difference between them is generated via current utterance. Hence, how to connect the previous SQL query and current utterance for generating corresponding SQL is important in context-dependent scenario. We use a hybrid pointer-generator network to copy a token from the previous SQL query or generate a new token from vocabulary during each decoding step. First, we use another bi-directional LSTM to encode the previous SQL query and define $\mathbf{h}_p^Q$ as the p^{th} hidden state of the previous predicted SQL query. Then, we calculate the attention between current decoder hidden state and the previous SQL query as follows:

$$\begin{aligned} s_k(p) &= (\mathbf{v}^{dp})^T \tanh(\mathbf{W}_1^{dp}\mathbf{h}_k^D + \mathbf{W}_2^{dp}\mathbf{h}_p^Q) \\ \alpha_k^{query} &= \text{softmax}(s) \\ \mathbf{c}_k^{query} &= \sum_{l=1}^{|q|} \alpha_k^{query}(p)\mathbf{h}_p^Q \ , \end{aligned} \tag{7}$$

where $|q|$ is the length of previous SQL query, $\mathbf{v}^{dp}$, $\mathbf{W}_1^{dp}$ and $\mathbf{W}_2^{dp}$ are learnable parameters. Finally, we modify the context vector as:

$$\mathbf{c}_k = [\mathbf{c}_k^{token}; \mathbf{c}_k^{schema}; \mathbf{c}_k^{query}] \ . \tag{8}$$

In addition, we generate a switch p_{copy} from context vector and decoder hidden state in step k:

$$p_{copy} = \sigma(\mathbf{c}_k\mathbf{W}_{copy}^1 + \mathbf{h}_k^D\mathbf{W}_{copy}^2), \tag{9}$$

where $\mathbf{W}_{copy}^1$ and $\mathbf{W}_{copy}^2$ are the learnable parameters, σ is the sigmoid function. p_{copy} is used to choose whether to copy a token from the previous SQL query via the previous SQL query attention distribution α_k^{query} or generate a new token. We define q_i as the i^{th} token in previous SQL query and modify the output probability distribution as follows:

$$P(y_k) = p_{copy}\sum\nolimits_{i:q_i=y_k} \alpha_k^{query}(i) + (1 - p_{copy})P_{vocab}(y_k). \tag{10}$$

3.5 Guide Component

The guide component in our model is utilized to avoid the table-column dependency and foreign key dependency errors. In order to solve misprediction of table-column dependencies, we design an intermediate state of the SQL query as shown in Figure 5 to predict tables first. Once tables are predicted, we prune the columns which do not belong to the predicted tables to avoid table-column dependency error. For avoiding the foreign key dependency error, we use a simple filter method to prune the tables which do not connect the predicted tables via foreign key constraints.

We apply guide component in the decoder, and use heuristic algorithm to get the prediction type in each decoding step. Once the prediction type is obtained, we modify $\mathbf{h}^C$ as follows:

$$\mathbf{h}^C = \begin{cases} \mathbf{h}_{table}^S, & \text{if } type \text{ is } table \\ \mathbf{h}_{column}^S, & \text{otherwise} \ , \end{cases} \tag{11}$$

Original: SELECT language, COUNT(*) FROM tv_channel GROUP BY language ORDER BY COUNT(*) ASC LIMIT 1

IntermediateState: FROM tv_channel SELECT language, COUNT(*) GROUP BY language ORDER BY COUNT(*) ASC LIMIT 1

Figure 5: An example of getting an intermediate state of the SQL query, and the FROM clause is moved to the first place in the SQL query.

Model	Question Matching	Interaction Matching
SyntaxSQL-con (Yu et al., 2019a)	13.8	2.1
CD-Seq2Seq (Yu et al., 2019a)	15.1	2.7
EditSQL(BERT) (Zhang et al., 2019)	39.9	12.3
PG-GSQL(BERT)	**41.2**	**16.4**

Table 1: Overall results of question matching and interaction matching accuracy on CoSQL dev set.

where $\mathbf{h}^S_{table}$ is that the schema embedding only contains table embeddings which link previous predicted tables via foreign key. If there is no predicted table, we use all table embeddings as $\mathbf{h}^S_{table}$. $\mathbf{h}^S_{column}$ only contains column embeddings which belong to the previous predicted tables.

3.6 BERT Enhanced Embedding

We employ BERT (Devlin et al., 2019) to augment the embeddings of utterances, schemas and previous SQL queries in our model. The method we use BERT is different from previous models (Hwang et al., 2019; Zhang et al., 2019), we only use the utterance tokens and table representations which are obtained from section 3.1 to the pretrained small cased BERT model. The sequence is fed into the pretrained BERT model as follows:

$$\texttt{[CLS]}, X_i, \texttt{[SEP]}, c_{11}, \ldots, c_{1m}, t_1, \texttt{[SEP]}, c_{21}, \ldots, t_{n-1}, \texttt{[SEP]}, c_{n1}, \ldots, c_{nm}, t_n, \texttt{[SEP]},$$

where [CLS] and [SEP] are split tokens, X_i is the utterance tokens, and c_{ij} is the j^{th} column in table t_i. We use the hidden states at the last layer in BERT as the token embeddings.

4 Experiment

In this section, we evaluate the effectiveness of our model on two large complex cross-domain context-dependent text-to-SQL datasets: SParC contains 3034, 421, 842 interactions and CoSQL contains 2164, 292, 551 interactions for training, development and testing. As the test sets of SParC and CoSQL are unreleased, we only evaluate our model on their dev sets. We evaluate our model using question matching accuracy (the exact set matching score over all questions) and interaction matching accuracy (the exact set matching score over all interactions). We do not use any extra data to boost our model for fair comparison. In addition, we conduct an ablation study to analyze the contribution of each innovation on SParC dev set.

4.1 Experimental Setup

Our implementation is based on PyTorch (Paszke et al., 2019) and we use Adam (Kingma and Ba, 2015) for optimization. The hyperparameter h is set to 5 in our model, and we use 50-dimensional position embeddings which are initialized from a random uniform distribution $U[0.1, 0.1]$ and are fixed during training. In addition, we use the pretrained 300-dimensional GloVe word embeddings (Pennington et al., 2014) for utterance embeddings, schema embeddings and keyword embeddings, the keyword embeddings are also fixed. The initial learning rate is 0.001 in PG-GSQL and it will be multiplied by 0.8 when the validation loss exceeds the previous epoch. In addition, when using BERT instead of GloVe, we set the learning rate of BERT to 1e-5. We use the official evaluation script[2] to calculate question matching accuracy and interaction matching accuracy.

[2]https://github.com/taoyds/sparc

Model	Question Matching	Interaction Matching
SyntaxSQL-con (Yu et al., 2019b)	18.5	4.3
CD-Seq2Seq (Yu et al., 2019b)	17.1	6.7
EditSQL (Zhang et al., 2019)	33.0	16.4
EditSQL(BERT) (Zhang et al., 2019)	47.2	29.5
PG-GSQL(BERT)	**53.1**	**34.7**
- pointer-generator network	41.2	15.2
- pointer-generator network - guide	35.8	12.8
PG-GSQL	34.0	19.0
- pointer-generator network	28.1	10.0
- pointer-generator network - guide	24.3	8.1

Table 2: Overall results of question matching and interaction matching accuracy on SParC dev set.

Model	Goal Difficulty			
	Easy (481)	Medium (441)	Hard (145)	Extra Hard (133)
SyntaxSQL-con (Yu et al., 2019b)	38.9	7.3	1.4	0.7
CD-Seq2Seq (Yu et al., 2019b)	35.1	7.0	2.8	0.8
PG-GSQL(BERT)	**72.1**	**50.1**	**32.4**	**16.5**
PG-GSQL	50.9	29.7	13.1	9.8

Table 3: Accuracy of question matching on SParC dev set in different hardness levels.

4.2 Results

Table 1 shows the comparisons of PG-GSQL with other models on CoSQL dev set, the performance of EditSQL on CoSQL is obtained from official page[3]. As illustrated, our model achieves 41.2% question matching accuracy and 16.4% interaction matching accuracy when using BERT augmentation, and it outperforms the EditSQL by 1.3% question matching accuracy and 4.1% interaction matching accuracy.

Table 2 shows the results of question matching accuracy and interaction matching accuracy by comparing PG-GSQL with previous models on SParC dataset. PG-GSQL outperforms EditSQL, which obtains 34.0% question matching accuracy and 19.0% interaction matching accuracy. When using BERT augmentation, PG-GSQL achieves 53.1% question matching accuracy and 34.7% interaction matching accuracy on the dev set. Compared with the previous state-of-the-art model EditSQL, our model improves question matching accuracy and interaction matching accuracy by 5.9% and 5.2%, respectively.

Furthermore, we also evaluate the performance of PG-GSQL in different hardness levels on SParC dev set according to the official classification. There are 481, 441, 145, 133 questions in easy, medium, hard and extra hard levels respectively. As shown in Table 3, compared to previous baseline models which provide the data in all four hardness levels, our model outperforms previous models by a large margin in all hardness levels. We further study the performance of PG-GSQL in different turns by official classification. There are 421, 421, 269, 89 questions in turn 1 to 4. As shown in Table 4.1, our model outperforms baseline models in all turns on dev set. In addition, we observe that utterances in the later turns have greater dependencies over previous turns and greater risks for error propagation.

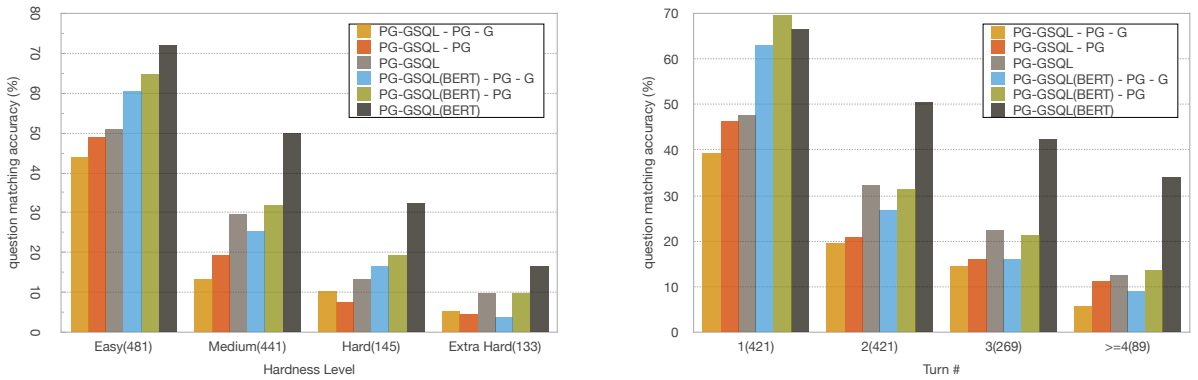
4.3 Ablation Study

We ablate the major novel innovations of PG-GSQL and PG-GSQL(BERT) to analyze their contributions on SParC dev set. Specifically, we first substitute the base decoder for the pointer-generator network and Table 2 shows that the performance drops by 5.9% in question matching accuracy and 9% in interaction

[3]https://github.com/ryanzhumich/editsql

Model	Turn #			
	1 (421)	2 (421)	3 (269)	>=4 (89)
SyntaxSQL-con (Yu et al., 2019b)	38.6	11.6	3.7	1.1
CD-Seq2Seq (Yu et al., 2019b)	31.4	12.1	7.8	2.2
PG-GSQL(BERT)	**66.5**	**50.6**	**42.4**	**34.1**
PG-GSQL	47.7	32.3	22.3	12.5

Table 4: Accuracy of question matching on SParC dev set in different turns.



(a) different hardness levels

(b) different turns

Figure 6: Effects of different innovations in PG-GSQL at different hardness levels and different turns on SParC dev set.

matching accuracy. When using BERT embeddings, it is obvious that interaction matching accuracy drops by a large margin of 19.5%. This shows hybrid pointer-generator network does effectively reuse the previous SQL query tokens. Then, we disable the guide component, which leads to the performance goes down by 3.8% in question matching accuracy and 1.9% in interaction matching accuracy. When incorporating BERT, Table 2 shows that guide component makes a significant drop in question matching accuracy to 35.8%. This demonstrates that the guide component is able to effectively avoid dependency errors during decoding phase.

In addition, to study the performance of PG-GSQL and PG-GSQL(BERT) in detail, we evaluate the effects of different innovations at different hardness levels and different turns as shown in Figure 6. Figure 6(a) shows that pointer-generator network makes a significant improvement in predicting complex SQL queries which occur in later turns as shown in Figure 6(b). Furthermore, Figure 6(b) more specifically illustrates that guide component improves the question matching accuracy in turn 1, and hybrid pointer-generator network depends on the turn 1 to promote the interaction matching accuracy.

5 Conclusion

In this paper, we present PG-GSQL with hybrid pointer-generator network and guide component to address the cross-domain and context-dependent text-to-SQL task. Experimental results show pointer-generator network is capable of reusing the previous SQL query tokens to significantly improve interaction matching accuracy. Furthermore, guide component avoids table-column dependency and foreign key dependency errors during decoding phase, and column-table linking improves the prediction accuracy of tables. The ablation study shows that our model improves the performance not only in predicting simple queries, but also in predicting nested, complex queries in unseen databases.

References

Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 1:29–81.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR 2015 : International Conference on Learning Representations 2015*.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *ACL 2019 : The 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565.

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

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *ACL 2018: 56th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 731–742.

Alessandra Giordani and Alessandro Moschitti. 2012. Generating sql queries using natural language syntactic dependencies and metadata. In *NLDB*.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *ACL 2019 : The 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The atis spoken language systems pilot corpus. *Proceedings of the workshop on Speech and Natural Language - HLT '90*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *CoRR*, abs/1902.01069.

Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *ICLR 2015 : International Conference on Learning Representations 2015*.

Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *very large data bases*, 8(1):73–84.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *COLING*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1073–1083.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *NAACL HLT 2018: 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 2238–2249.

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive sql queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, volume 52, pages 452–466.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. *CoRR*, abs/1911.04942.

David H. D. Warren and Fernando Pereira. 1981. An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8:110–122.

Xiaojun Xu, Chang Liu, and Dawn Xiaodong Song. 2018. Sqlnet: Generating structured queries from natural language without reinforcement learning. *ArXiv*, abs/1711.04436.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: query synthesis from natural language. *PACMPL*, 1:63:1–63:26.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir R. Radev. 2018b. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. In *EMNLP*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018c. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *2019 Conference on Empirical Methods in Natural Language Processing*, pages 1962–1979.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Irene Li Heyang Er, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Vincent Zhang Jonathan Kraft, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019b. Sparc: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy. Association for Computational Linguistics.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based sql query generation for cross-domain context-dependent questions. In *2019 Conference on Empirical Methods in Natural Language Processing*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.