RECPARSER: A Recursive Semantic Parsing Framework for Text-to-SQL Task

Yu Zeng[1*], **Yan Gao**[2], **Jiaqi Guo**[3*], **Bei Chen**[2], **Qian Liu**[4*], **Jian-Guang Lou**[2], **Fei Teng**[†1] and **Dongmei Zhang**[2]
[1]Southwest Jiaotong University, Chengdu, China
[2]Microsoft Research Asia, Beijing, China
[3]Xi'an Jiaotong University, Xi'an, China
[4]Beihang University, Beijing, China
zengyutk@my.swjtu.edu.cn, {yan.gao, beichen, jlou, dongmeiz}@microsoft.com,
jasperguo2013@stu.xjtu.edu.cn, qian.liu@buaa.edu.cn, fteng@swjtu.edu.cn

Abstract

Neural semantic parsers usually fail to parse long and complicated utterances into nested SQL queries, due to the large search space. In this paper, we propose a novel recursive semantic parsing framework called RECPARSER to generate the nested SQL query layer-by-layer. It decomposes the complicated nested SQL query generation problem into several progressive non-nested SQL query generation problems. Furthermore, we propose a novel Question Decomposer module to explicitly encourage RECPARSER to focus on different components of an utterance when predicting SQL queries of different layers. Experiments on the Spider dataset show that our approach is more effective compared to the previous works at predicting the nested SQL queries. In addition, we achieve an overall accuracy that is comparable with state-of-the-art approaches.

1 Introduction

Text-to-SQL is one of the most important sub-tasks of semantic parsing in natural language processing (NLP). It maps natural language utterances to corresponding SQL queries. By helping non-experts to interact with ever increasing databases, the task has many important potential applications in real life, and thus receives a great deal of interest from both industry and academia [Li and Jagadish, 2016; Zhong *et al.*, 2017; Affolter *et al.*, 2019].

Composing nested SQL queries is a challenging problem in Text-to-SQL. Several works [Guo *et al.*, 2019; Zhang *et al.*, 2019b; Bogin *et al.*, 2019; Yu *et al.*, 2018a] have attempted to deal with this problem on the recently proposed dataset Spider [Yu *et al.*, 2018b], which contains nested SQL queries over different databases with multiple tables. However, due to the large search space, existing neural semantic parsers do not perform well on composing nested SQL queries [Zhang *et al.*, 2019a; Affolter *et al.*, 2019].

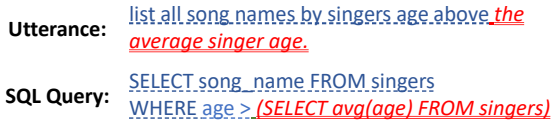

Figure 1: An example of the component mapping relationship between the nested SQL query and its corresponding utterance.

In this paper, we focus on the problem of composing nested SQL queries in Text-to-SQL task. Firstly, as we know the nested SQL query representation is formalized as a kind of recursive structure, thus a nested SQL query generation problem can be decomposed into several progressive non-nested SQL query generation problems by nature. Furthermore, we found that there is a strong component mapping relationship between the nested SQL query and its corresponding utterance. Figure 1 shows an example of the component mapping relationship between the nested SQL query and its corresponding utterance. We see that, for this nested SQL query, the outside layer "SELECT song_name FROM singers WHERE age >" and the inside layer "SELECT avg(age) FROM singers" correspond to different utterance components "list all song names by singers age above" and "the average singer age" respectively.

Inspired by these observations, we propose a recursive semantic parsing framework called RECPARSER to generate the nested SQL query layer-by-layer from outside to inside. Furthermore, we propose a novel Question Decomposer module to explicitly encourage RECPARSER to focus on different parts of an utterance when predicting SQL queries in different layers. RECPARSER consists of four modules: Initial Encoder, Iterative Encoder, Question Decomposer and SQL Generator. In the initialization stage, we encode the user utterance and the database schema with the Initial Encoder. Then, by recursively calling the Question Decomposer, the Iterative Encoder and the SQL Generator modules, we generate the nested SQL query layer-by-layer. Finally, we compose SQL queries of different layers into the whole SQL query when encountering a recursive termination condition. Concretely, in each recursion round, the Question Decomposer updates the utterance representation with a *soft-attention-mask* mechanism and a *recursive-attention-divergency* loss

[*]Work done during an internship at Microsoft Research.
[†]Corresponding Author

proposed by us for the next recursion; the Iterative Encoder contextualizes the schema representation with the updated utterance representation; the SQL Generator takes the contextualized schema representation as input and generates the non-nested SQL query of the current layer. Our SQL Generator is a simple yet effective multi-task classification model.

We argue that RECPARSER has two advantages. Firstly, in this divide-and-conquer faction, we simplify the difficulty of the nested query generation problem. Our SQL Generator only learns to generate non-nested SQL queries instead of the whole nested SQL query, thus largely reducing the search space and alleviating training difficulties. Secondly, through updating the utterance representation recursively with the *soft-attention-mask* mechanism and the *recursive-attention-divergency* loss, RECPARSER can focus on different parts of an utterance when predicting different SQL query layers, thus minimizing the interference of irrelevant parts.

On the Spider benchmark [Yu *et al.*, 2018b], RECPARSER achieves a state-of-the-art 39.3% accuracy on the nested SQL query and a comparable 54.3% accuracy on the overall SQL query. When augmented with BERT [Devlin *et al.*, 2018], RECPARSER reaches up to a state-of-the-art 46.8% accuracy on the nested SQL query and a state-of-the-art 63.1% accuracy on the overall SQL query. Our contributions are summarized as follows.

- We propose a recursive semantic parsing framework called RECPARSER to decompose a complicated nested SQL query generation problem into several progressive non-nested SQL query generation problems.
- We propose a novel Question Decomposer module with a *soft-attention-mask* mechanism and a *recursive-attention-divergency* loss to explicitly encourage RECPARSER to focus on different parts of the utterance when predicting SQL queries in different layers.
- Our approach is more effective compared to previous works at predicting nested SQL queries. In addition, we achieve an overall accuracy that is comparable with state-of-the-art approaches.

2 Related Works

Composing Nested SQL Query. The problem of composing nested SQL queries has been studied for decades [Androutsopoulos *et al.*, 1995]. Most of the early proposed systems are rule-based [Popescu *et al.*, 2003; Li and Jagadish, 2014]. Recently, with the development of advanced neural approaches and the release of the complex and cross-domain Text-to-SQL dataset Spider [Yu *et al.*, 2018b], several neural semantic parsers [Guo *et al.*, 2019; Zhang *et al.*, 2019b; Bogin *et al.*, 2019; Lee, 2019] have attempted to generate the nested SQL queries with a complicated grammar-based decoder. However, applying grammar-based decoders to general programming languages such as SQL query is very challenging [Lin *et al.*, 2019] and difficult to generalize to other semantic parsing tasks. Also, Finegan-Dollak *et al.* [2018] shows that the sequence-to-tree approach is inefficient when generating complicated SQL queries from an utterance. Our work differs from them in our use of a divide-and-conquer generation procedure.

Recursive Mechanism in Semantic Parsing. Recursive mechanism has been successfully used in complicated semantic parsing tasks [Andreas *et al.*, 2016; Rabinovich *et al.*, 2017]. Recently, several works [Yu *et al.*, 2018a; Lee, 2019] also employ the recursive mechanism to generate nested SQL queries in Text-to-SQL task. SyntaxSQL-Net [Yu *et al.*, 2018a] employs a SQL specific syntax tree-based decoder that calls a collection of recursive modules for decoding. RCSQL [Lee, 2019] proposes a SQL clause-wise decoding architecture. It recursively calls different SQL clause decoders to predict nested SQL queries. Compared with their approaches, we recursively call both encoder module and SQL generator module for each recursion, while they only call a set of SQL generator modules. In addition, we propose a novel Question Decomposer module to capture the mapping relationship between different SQL query layers and their corresponding components in utterance.

Decomposing Complicated Question. Decomposing complicated question is widely used in many semantic parsing works [Iyyer *et al.*, 2016; Talmor and Berant, 2018; Zhang *et al.*, 2019a]. They utilize the decompositionality of complicated questions to help question understanding. Inspired by them, we use the *soft-attention-mask* mechanism and the *recursive-attention-divergency* loss to model the decomposing process of the utterance and update the utterance representation for next recursion round. In this way, we don't need to make hard decisions and it could be learned in an end-to-end fashion without extra labeling work.

Multi-Task Learning in Text-to-SQL Task. In multi-task learning, the Text-to-SQL task is decomposed into several sub-tasks, each predicting a part of the final SQL program. Compared with sequence-to-sequence-style models, multi-task learning does not require the SQL queries to be serialized and thus avoid the "ordering issue" [Xu *et al.*, 2018]. Existing state-of-the-art multi-task learning methods [Hwang *et al.*, 2019; He *et al.*, 2019] have already surpassed human performance on WikiSQL [Zhong *et al.*, 2017] dataset. However, existing methods are limited to the specific SQL sketch of WikiSQL, which only supports very simple queries. In this work, we propose a simple yet effective multi-task classification model to generate arbitrary non-nested SQL query as the SQL Generator module in our framework.

3 Methodology

In this section, we will describe our RECPARSER framework in detail. As described in Figure 2, RECPARSER consists of four modules: Initial Encoder, Iterative Encoder, Question Decomposer and SQL Generator. At first, we get the initial representation of utterance and database schema with the Initial Encoder. Then, by recursively calling the Iterative Encoder, the Question Decomposer, and the SQL Generator, we generate the nested SQL query layer-by-layer from outside to inside. Finally, we compose the SQL queries of different layers into the whole SQL query when encountering a recursive termination condition.

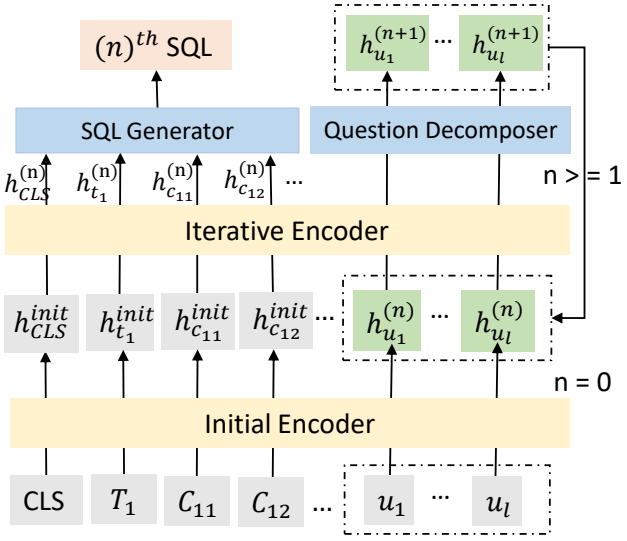

Figure 2: RECPARSER Architecture.

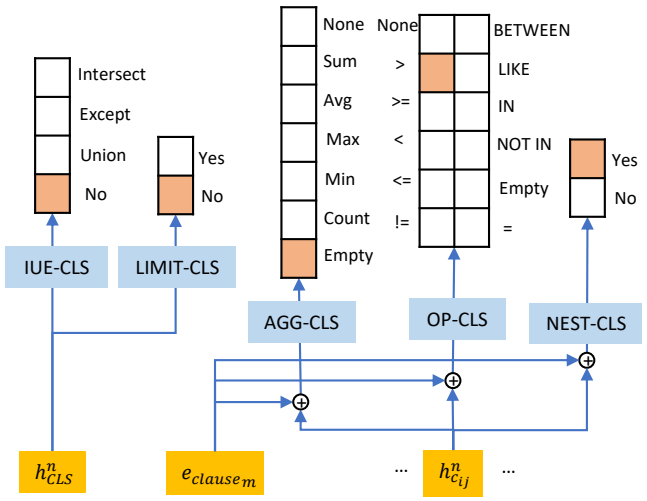

Figure 3: An overview of the SQL Generator.

3.1 Initial Encoder

Given a user input utterance U and its corresponding database schema S, the Initial Encoder generates the initial representation h_U^{init} and h_S^{init}. Let $U = (u_1, .., u_k, .., u_l)$ and $S = \{t_1, c_{11}, ..., t_i, c_{ij}, ...\}$, where u_k denotes the k-th utterance token, l is the length of utterance, t_i denotes the i-th table name in database schema, and c_{ij} denotes the j-th column name in the i-th table. Following Hwang *et al.* [2019], we first concatenate the utterance U and the database schema S as a single sequence separated by token $[CLS]$, $[TAB]$, $[COL]$, and $[SEP]$ as follows.

$$[CLS], [TAB], t_1, [COL], c_{11}.., [COL], c_{ij}..[SEP], u_1..u_l, \quad (1)$$

where we put $[CLS]$ token at the beginning of the sequence to capture the contextualized representation of the whole sequence. Then we employ a Bidirectional LSTM [Hochreiter and Schmidhuber, 1997] to get the h_U^{init} and the h_S^{init}, where $h_U^{init} = (h_{u_1}, .., h_{u_k}, .., h_{u_l})$ and $h_S^{init} = (h_{t_1}..., h_{t_i}, h_{col_{ij}}, ...)$. Concretely, for each table and column in database schema, we use the representation of the separator $[TAB]$ and $[COL]$ before each table and column as their corresponding representations. Moreover, we use h_{CLS} denotes the representation of token $[CLS]$. By concatenating the utterance and the database schema together, we capture the relationship between the utterance and the corresponding database schema and get the contextualized representations.

3.2 Iterative Encoder

Let n denote the n-th recursion round. Given $h_U^{(n)}$, h_{CLS}^{init} and h_S^{init} as inputs, the goal of the Iterative Encoder is to generate the database schema representation $h_S^{(n)}$ and the $[CLS]$ representation $h_{CLS}^{(n)}$ of the n-th recursion round. Then, $h_S^{(n)}$ and $h_{CLS}^{(n)}$ will be used as the inputs of the SQL Generator to generate the current layer SQL query. Note that if it is the 0-th recursion round, we use h_U^{init} as h_U^0. Specifically, following Hwang *et al.* [2019], we concatenate h_{CLS}^{init}, h_S^{init} and $h_U^{(n)}$ in a single representation $h^{(n)}$. Then we apply self-attention mechanism [Vaswani *et al.*, 2017] to update $h^{(n)}$ as follows.

$$\begin{aligned} \alpha^{(n)} &= \text{softmax}(\frac{QK^T}{\sqrt{d}}), \\ h^{(n)} &= \alpha^{(n)} V, \end{aligned} \quad (2)$$

where $Q = w_q h^{(n)}$, $K = w_k h^{(n)}$, $V = w_v h^{(n)}$ and $w_q \in \mathbb{R}^{d \times d}, w_k \in \mathbb{R}^{d \times d}, w_v \in \mathbb{R}^{d \times d}$ are trainable parameters.

3.3 SQL Generator

Given the database schema representation $h_S^{(n)}$ and the $[CLS]$ representation $h_{CLS}^{(n)}$, the goal of the SQL Generator is to generate the non-nested SQL query at the current recursion round n. According to the syntactical grammar of the SQL query, we transform the non-nested SQL query generation problem to a slot-filling problem. Concretely, we design a local template that indicates the property of each column in each SQL clause (e.g., SELECT clause) and a global template that indicates the property of the SQL query.

Firstly, we generate $N \times M$ local templates where N is the number of columns in the given database schema S and M is the number of SQL clauses. Each template has three local slots, i.e., Aggregator, Operator and Nest. Notice that the slot Nest is used to indicate whether the column in corresponding clause has nested SQL query or not. Figure 3 presents all candidates of the Aggregator, the Operator and the Nest. Notice that EMPTY means that the column is not in the corresponding SQL clause. As shown in Figure 3, they are predicted by three classifiers, i.e., AGG-CLS, OP-CLS and NES-CLS respectively. Taking the OP-CLS as an example, the probability of the operator op for col_{ij} in $clause_m$ is computed as following:

$$\begin{aligned} h'^{(n)}_{col_{ij}} &= h^{(n)}_{col_{ij}} + e_{clause_m}, \\ P_{op}^{(col_{ij},m)} &= \text{softmax}(W_{op} h'^{(n)}_{col_{ij}} + b_{op}), \end{aligned} \quad (3)$$

where e_{clause_m} is the m-th clause embedding. $W_{op} \in \mathbb{R}^{n_{op} \times d}$, $b_{op} \in \mathbb{R}^{n_{op}}$, $e_{clause_m} \in \mathbb{R}^d$ are trainable parameters

and n_{op} is the number of operator types. The same way can be also applied to the AGG-CLS and NES-CLS to calculate $P_{agg}^{(col_{ij},m)}$ and $P_{nes}^{(col_{ij},m)}$. Specially, we have 7 kinds of SQL clauses (i.e., SELECT, WHERE_AND, WHERE_OR, ORDER_ASC, ORDER_DES, GROUP, and HAVING), in which the original WHERE clause is divided into WHERE_AND and WHERE_OR to distinguish two different connectors (i.e., AND and OR) in the WHERE clause. Similarly, the ORDER clause is also divided into ORDER_ASC and ORDER_DES to distinguish two different sort types (i.e., DES and ASC).

Next, we generate one global template which has two global slots, i.e., Limit and IUE. Limit indicates if the SQL query contains a keyword LIMIT. IUE indicates whether the SQL query contains structures such as Intersect, Union, Except or not. We use another two classifiers, i.e., LIMIT-CLS and IUE-CLS to predict them respectively. In IUE-CLS, the probability of the candidate iue is calculated as follows.

$$P_{iue} = \text{softmax}(W_i h_{CLS}^{(n)} + b_i), \tag{4}$$

where $W_i \in \mathbb{R}^{n_i \times d}$, $b_i \in \mathbb{R}^{n_i}$ are trainable parameters, n_i is the number of IUE types. P_{limit} is calculated in the same way by LIMIT-CLS. After filling all the slots of all the templates, we use a heuristic method to compose these templates to a SQL query according to the SQL grammar.

At last, after predicting all the other clauses, similar to [Lee, 2019], we use a heuristics to generate the FROM clause of the current SQL query layer. We first collect all the columns that appear in the predicted SQL, and then we JOIN tables that include these predicted columns.

3.4 Question Decomposer

The goal of the Question Decomposer is to update the utterance representation for the next recursion round. Intuitively, we hope that RECPARSER could learn to decompose the utterance layer-by-layer and focus on different components of an utterance when predicting SQL queries of different layers. We achieve this target by two simple yet effective methods: (1) **Soft Attention Mask.** Inspired by Kim *et al.* [2018] which leverages a soft attention mask mechanism to do feature selection, we propose a novel method to soft mask the parts of utterance that has already been focused by previous recursion rounds. (2) **Recursive Attention Divergence Loss.** Inspired by Wei *et al.* [2019] which uses JS divergence loss to minimize the attention divergence in code generation task, we propose a novel regularization loss to encourage the model to maximize the attention divergence between the adjacent recursion rounds. In this way, we explicitly encourage RECPARSER to focus on different components of an utterance when predicting different SQL query layers, thus minimizing the interference of irrelevant components of the utterance.

Soft Attention Mask

Let $\alpha_{u_i}^{(n)}$ denotes the attention weight of token $[CLS]$ on i-th token of utterance U, which is produced by Equation 2. Intuitively, to help the model focus on different components of an utterance in the next recursion, we want to mask the component that has been focused on the current recursion. Instead of directly masking the tokens of the utterance, we generate the utterance representation of next recursion by a soft-attention-mask method as follows.

$$\begin{aligned} E_{mask_i}^{(n)} &= \alpha_{u_i}^{(n)} W_{mask}, \\ h_{u_i}^{(n+1)} &= W_U h_{u_i}^{(n)} + W_m E_{mask_i}^{(n)}, \end{aligned} \tag{5}$$

where $E_{mask_i}^{(n)} \in \mathbb{R}^d$ is the soft mask embedding, $W_{mask} \in \mathbb{R}^d, W_U \in \mathbb{R}^{d \times d}, W_m \in \mathbb{R}^{d \times d}$ are trainable parameters.

Recursive Attention Divergence Loss

To encourage the model to focus on different components of an utterance in each recursion round, we design the Recursive Attention Divergence Loss $\mathcal{L}_{div}$ to regularize the attention divergence between the attention weights of the adjacent recursion rounds.

Let $\alpha_U^{(n)}$ and $\alpha_U^{(n-1)}$ denote the attention weights of the adjacent recursion rounds for a same utterance. We apply the negative Jensen–Shannon divergence [Fuglede and Topsoe, 2004], a symmetric measurement of similarity between two probability distributions, to maximize the distance between these two attention weights. Let P denotes the $\alpha_U^{(n)}$ and Q denotes the $\alpha_U^{(n-1)}$:

$$\mathcal{L}_{div} = -\frac{1}{2}(D_{KL}(P||\frac{P+Q}{2}) + D_{KL}(Q||\frac{P+Q}{2})), \tag{6}$$

where D_{KL} is the Kullback–Leibler divergence, defined as $D_{KL}(p||q) = \sum_x p(x) \log(\frac{p(x)}{q(x)})$ which measures how one probability distribution q diverges from another probability distribution p.

3.5 Composing the Final SQL Query

The final SQL query is obtained through backtracking. The SQL Generator will return a SQL query segment after the end of the recursive calling. The recursive termination condition is that the IUE slot and all NEST slots are NO. Specifically, if the slot NEST is YES, we return a SQL query with a placeholder token [SUBQUERY] after its corresponding column that is in its clause (e.g., SELECT name FROM singers WHERE age > [SUBQUERY] and sex = 'male'). Similarly, if the IUE slot is not NO, we put corresponding placeholder token after the returned SQL query result (e.g., SELECT name FROM singers [EXCEPT]). Finally, we construct the final SQL by replacing the placeholder token with the returned SQL query segment.

3.6 Loss Function

The loss function of RECPARSER is defined as follows.

$$\begin{aligned} \mathcal{L} = \lambda_{AGG}\mathcal{L}_{AGG} + \lambda_{OP}\mathcal{L}_{OP} + \lambda_{NES}\mathcal{L}_{NES} \\ + \lambda_{IUE}\mathcal{L}_{IUE} + \lambda_{LIMIT}\mathcal{L}_{LIMIT} + \lambda_{div}\mathcal{L}_{div}, \end{aligned} \tag{7}$$

where $\mathcal{L}_{AGG}$, $\mathcal{L}_{OP}$, $\mathcal{L}_{NES}$, $\mathcal{L}_{IUE}$, $\mathcal{L}_{LIMIT}$ are the loss functions of AGG-CLS, OP-CLS, NES-CLS, IUE-CLS and LIMIT-CLS in the SQL Generator respectively. All loss functions of the classifiers are cross-entropy loss. $\mathcal{L}_{div}$ is the Recursive Attention Divergence Loss described in Equation 6. λ_{AGG}, λ_{OP}, λ_{NES}, λ_{IUE}, λ_{LIMIT}, λ_{div} are loss weights and all of them are hyper parameters.

Approach	Accuracy	Accuracy (BERT)
SyntaxSQLNet	0	-
Global-GNN	33.0%	-
EditSQL	-	40.4%
IRNET	34.2%	41.8%
RECPARSER(no DB)	**38.0%**	**46.8%**
RECPARSER	**39.3%**	**46.8%**

Table 1: Nested SQL Exact Matching accuracy on Spider development set. "no DB" means the database content is not used; "BERT" means the results are obtained with BERT enhanced approaches and "-" means the methods are not proposed in corresponding settings, e.g., with or without BERT.

Approach	Accuracy
SyntaxSQLNet	24.8%
RCSQL	28.5%
Global-GNN	52.7%
IRNET	53.2%
RECPARSER	**54.3%**
Edit-SQL(BERT)	57.6%
RECPARSER(BERT, no DB)	60.5%
IRNET(BERT)	61.9%
RECPARSER(BERT)	**63.1%**

Table 2: SQL Exact Matching accuracy on Spider development set

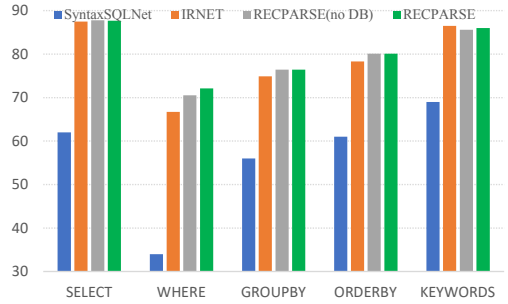

Figure 4: F1 scores of Component Matching accuracy of SyntaxSQLNet, IRNET, RECPARSER(no DB) and RECPARSER on Spider development set. All methods are augmented with BERT.

4 Experiments

In this section, we evaluate the effectiveness of RECPARSER on both nested SQL queries and all SQL queries by comparing it to state-of-the-art approaches and ablating several design choices in RECPARSER to understand their contributions.

4.1 Experiment Setup

Dataset. We conduct our experiments on Spider [Yu *et al.*, 2018b], a large-scale, human-annotated and cross-domain Text-to-SQL benchmark, which contains 7,000/1,034 question-SQL query pairs in train and development set[1]. We using SQL Exact Matching and Component Matching accuracy metrics proposed by Yu *et al.* [2018b] to evaluate RECPARSER and other approaches.

Baselines. We use the following methods for comparison evaluation: SyntaxSQLNet [Yu *et al.*, 2018a], RCSQL [Lee, 2019], Global GNN [Bogin *et al.*, 2019], EditSQL [Zhang *et al.*, 2019b], and IRNET [Guo *et al.*, 2019]. All of them employ an encoder-decoder architecture, but with different design choices: SyntaxSQLNet is a sequence-to-set model with a SQL specific syntax tree-based decoder; RCSQL employs a SQL clause-wise decoding network with recursive mechanism; Global GNN reasons over the DB structure and questions to make global decision; EditSQL proposes an editing mechanism-based network; and IRNET uses schema linking information and predicts an intermediate representation instead of the SQL query. Note that SyntaxSQLNet, RCSQL, EditSQL and IRNet do not use the database content, while Global GNN and RECPARSER use the database content.

Implementations. We implement RECPARSER with PyTorch. The dropout rate is 0.2. We use Adam with 1e-3 learning rate for optimization. Batch size is 64. Word embeddings are initialized with Glove. The dimensions of word embedding, type embedding and hidden vectors are set to 300. λ_{AGG}, λ_{OP}, λ_{NES}, λ_{IUE}, λ_{LIMIT}, λ_{div} in Equation 7 are set as 1, 1, 1, 0.1, 0.1, and 0.2 respectively. We use the database content to find the column whose value is exactly mentioned in the utterance and put a special token [*VALUE*] in front of that column.

[1]We don't use the test set of Spider, since the test set is blind to us and we can't obtain the nested SQL query of the test set to evaluate the effectiveness of our approach.

BERT. Language model pre-training has shown to be effective for learning universal language representations. To further study the effectiveness of our approach, we leverage BERT [Devlin *et al.*, 2018] to encode question and database schema to replace the bi-directional LSTM in the Initial Encoder. Our approach with BERT-base is denoted as RECPARSER(BERT).

4.2 Model Comparison

Table 1 presents the nested SQL Exact Matching accuracy of RECPARSER and various baselines on the development set. The result shows that RECPARSER clearly outperforms all the baselines with or without DB content used on the nested SQL query accuracy. Comparing to the methods that use various techniques to predict nested SQL query, such as complicated intermediate representation (e.g., IRNet) and other grammar-based decoders (e.g., EditSQL and Global-GNN), RECPARSER achieves a prominent improvement by employing a novel recursive generation framework. For example, it obtains 5.0% absolute improvement than the state-of-the-art method IRNET when both of them incorporate with BERT.

Next, we evaluate the overall SQL Exact Matching accuracy of RECPARSER and various baselines on the development set of Spider. As shown in Table 2, RECPARSER(BERT, no DB) achieves a 60.5% Exact Matching accuracy without using any rule-based schema linking techniques or intermediate representation that are used in IRNet(BERT). It demonstrates that RECPARSER(BERT, no DB) obtains a comparable overall accuracy than the state-of-the-art approaches at the same setting with min-

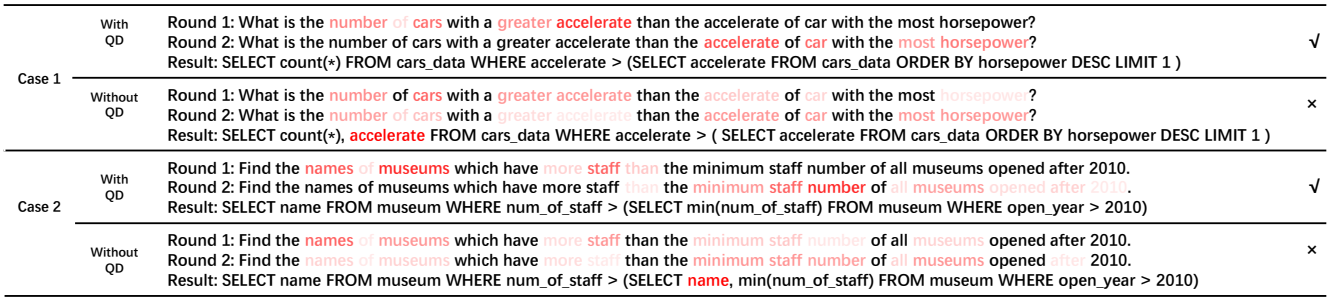

Figure 5: Two examples where Question Decomposer (QD) for RECPARSER leads to correct prediction. In the first example, RECPARSER without QD selects the wrong column in the outside SQL layer which belongs to the inner SQL layer. In the second example, RECPARSER without QD selects the wrong column in the inner SQL layer which belongs to the outside SQL layer. Color intensity in red reflects the attention weight, and the tokens in black have negligibly small weights.

Base-framework	57.4%
+ QD (- RAD loss)	58.5%
+ QD	60.8%
+ QD + DB	63.1%

Table 3: Ablation study results. Base-framework doesn't use the Question Decomposer (QD) and the database content (DB). 'QD (- RAD loss)' means that the QD does not use the recursive-attention-divergency loss.

imum efforts. When incorporating the database content, RECPARSER(BERT) gets a 63.1% Exact Matching accuracy and obtains a 1.2% absolute improvement over state-of-the-art method. It demonstrates the effectiveness of RECPARSER(BERT).

To further study the performance of RECPARSER in detail, following Yu *et al.* [2018a], we measure the average F1 score on different SQL query components on the development set. Here, we take BERT enhanced methods as an example. As shown in Figure 4, RECPARSER outperforms SyntaxSQLNet and IRNET on all components except KEYWORDS. Importantly, because of the good performance of RECPARSER on nested SQL query which belongs to the WHERE clause, there is a 3.4% absolute improvement compared to IRNET in WHERE clause. When incorporating with database content, the improvement is boosted to 5.4%.

4.3 Ablation Study

We conduct ablation study on RECPARSER(BERT) to analyze the contribution of each design choice. In detail, we first evaluate a base framework that does not use the Question Decomposer (QD) and the database content (DB). Then, we gradually apply each component to the base model. The ablation study is conducted on the development set. Table 3 shows the results of ablation study.

Firstly, it is clear that our base model significantly outperforms the well designed complicated architecture methods (e.g., Global-GNN and SyntaxSQLNet) by using the recursive framework with the simple yet effective multi-task SQL Generator module.

Secondly, using the Question Decomposer without the recursive-attention-divergency loss improves the performance of RECPARSER about 1.1%. It demonstrates the effectiveness of using the soft-mask-attention to update the utterance representation.

Thirdly, adding the recursive-attention-divergency loss further brings a 2.3% improvement. It demonstrates the effectiveness of our loss function. We observe that for those nested SQL queries, the base framework has the problem that tends to put the columns of the outside layer to the inside layer. The number of examples suffering from this problem decreases by 53% when using the Question Decomposer. It means that the Question Decomposer helps to minimize the interference between different SQL layers. At last, using the database content helps to recognize the value mentioned in the utterance, which brings a 2.3% improvement of performance.

4.4 Qualitative Analysis

Here we conduct a qualitative analysis on the effectiveness of the Question Decomposer (QD). Firstly, Figure 5 visualizes the attention weights α_U from Equation 2 in different rounds of the two examples, in which color intensity reflects the attention weight. As we can see, when incorporating the QD module, RECPARSER is much easier to focus on different components of the utterance between different rounds than removing the QD module. Secondly, according to the prediction results of the two examples, when removing the QD module, RECPARSER tends to be confused about the correct layer position of the selected column. When incorporating with the QD module, both of the two examples get the correct SQL queries. It demonstrates that the QD module help RECPARSER to minimize the interference between different utterance components that correspond to different SQL query layers.

5 Conclusion

In this paper, we propose a novel recursive semantic parsing framework called RECPARSER to decompose the complicated nested SQL generation problem into several progressive non-nested SQL query generation problems. Experiments on Spider dataset show that our approach is more effective compared to previous work at predicting the nested SQL queries. In addition, we achieve a comparable overall accuracy to state-of-the-art approaches.

References

[Affolter *et al.*, 2019] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28:793 – 819, 2019.

[Andreas *et al.*, 2016] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *NAACL*, pages 1545–1554, San Diego, California, June 2016. Association for Computational Linguistics.

[Androutsopoulos *et al.*, 1995] Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81, 1995.

[Bogin *et al.*, 2019] Ben Bogin, Matt Gardner, and Jonathan Berant. Global reasoning over database structures for text-to-sql parsing. In *EMNLP/IJCNLP*, 2019.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Finegan-Dollak *et al.*, 2018] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-SQL evaluation methodology. In *ACL*, pages 351–360, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[Fuglede and Topsoe, 2004] Bent Fuglede and Flemming Topsoe. Jensen-shannon divergence and hilbert space embedding. In *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, page 31. IEEE, 2004.

[Guo *et al.*, 2019] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. In *ACL*, 2019.

[He *et al.*, 2019] Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. X-sql: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*, 2019.

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

[Hwang *et al.*, 2019] Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*, 2019.

[Iyyer *et al.*, 2016] Mohit Iyyer, Wen tau Yih, and Ming-Wei Chang. Answering complicated question intents expressed in decomposed question sequences. *ArXiv*, abs/1611.01242, 2016.

[Kim *et al.*, 2018] Wonsik Kim, Bhavya Goyal, Kunal Chawla, Jungmin Lee, and Keunjoo Kwon. Attention-based ensemble for deep metric learning. In *ECCV*, pages 736–751, 2018.

[Lee, 2019] Dongjun Lee. Clause-wise and recursive decoding for complex and cross-domain text-to-sql generation. In *EMNLP/IJCNLP*, pages 6047–6053, 2019.

[Li and Jagadish, 2014] Fei Li and HV Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.

[Li and Jagadish, 2016] Fei Li and H. V. Jagadish. Understanding natural language queries over relational databases. *SIGMOD Record*, 45:6–13, 2016.

[Lin *et al.*, 2019] Kevin Lin, Ben Bogin, Mark Neumann, Jonathan Berant, and Matt Gardner. Grammar-based neural text-to-sql generation. *arXiv preprint arXiv:1905.13326*, 2019.

[Popescu *et al.*, 2003] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM, 2003.

[Rabinovich *et al.*, 2017] Maxim Rabinovich, Mitchell Stern, and Dan Klein. Abstract syntax networks for code generation and semantic parsing. In *ACL*, pages 1139–1149, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[Talmor and Berant, 2018] Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *NAACL*, pages 641–651, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[Wei *et al.*, 2019] Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. Code generation as a dual task of code summarization. In *NIPS*, pages 6559–6569, 2019.

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

[Yu *et al.*, 2018a] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *EMNLP*, pages 1653–1663, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

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

[Zhang *et al.*, 2019a] Haoyu Zhang, Jingjing Cai, Jianjun Xu, and Ji Wang. Complex question decomposition for semantic parsing. In *ACL*, 2019.

[Zhang *et al.*, 2019b] Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir R. Radev. Editing-based sql query generation for cross-domain context-dependent questions. In *EMNLP/IJCNLP*, 2019.

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