IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 7, 2019, accepted July 21, 2019, date of publication July 26, 2019, date of current version August 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2931464

Hierarchical Schema Representation for Text-to-SQL Parsing With Decomposing Decoding

MEINA SONG, (Member, IEEE), ZECHENG ZHAN, AND HAIHONG E.
School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China
Corresponding author: Meina Song (mnsong@bupt.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2018YFB1403003.

ABSTRACT Most of existing studies on parsing natural language (NL) for constructing structured query language (SQL) do not consider the complex structure of database schema and the gap between NL and SQL query. In this paper, we propose a schema-aware neural network with decomposing architecture, namely HSRNet, which aims to address the complex and cross-domain Text-to-SQL generation task. The HSRNet models the relationship of the database schema with a hierarchical schema graph and employs a graph network to encode the information into sentence representation. Instead of end-to-end generation, the HSRNet decomposes the generation process into three phases. Given an input question and schema, we first choose the column candidates and generate the sketch grammar of the SQL query. Then, a detail completion module fills the details based on the column candidates and the corresponding sketch. We demonstrate the effectiveness of our hierarchical schema representation by incorporating the information into different baselines. We further show that the decomposing architecture significantly improves the performance of our model. Evaluation of Spider benchmark shows that the hierarchical schema representation and decomposing architecture improves our parser result by 14.5% and 4.3% respectively.

INDEX TERMS Semantic parsing, SQL generation, deep learning, neural network, graph encoder, natural language process.

I. INTRODUCTION

Text-to-SQL, i.e., synthesizing SQL from natural language questions and query relational databases, has recently received renewed interest. Evaluations on public Text-to-SQL benchmarks, e.g., ATIS, GeoQuery and WikiSQL, have demonstrated the effectiveness of the existing SQL synthesizing approaches (beyond 85% exact matching accuracy) [1]–[6]. However, these approaches do not perform well on a newly released Text-to-SQL benchmark called, Spider [8]. Specifically, the recent neural approach [9] achieves 27% exact matching accuracy on Spider, which is much lower than that on other benchmarks. Unlike the ATIS, GeoQuery and Restaurants benchmarks, Spider focuses on the cross-domain settings assuming no database overlaps across the training and test sets. In comparison to WikiSQL [1], Spider contains databases with multiple tables and is far more complex in terms of the compositionality of SQL components and input question. In other words, Spider is more representative of the real-world synthesizing settings.

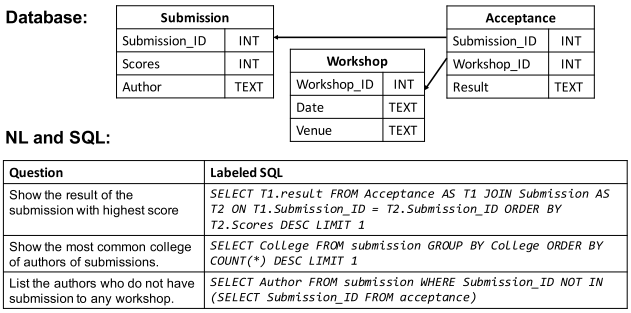

FIGURE 1. Example input questions x, SQL queries y_{sql} and corresponding database schema s. The first example shows the complex of schema and the others illustrate the complexity of labeled SQL components.

Spider benchmark brings new challenges leading to unsatisfactory performances of the existing parsing approaches, as shown in Fig. 1. Firstly, previous studies either had simple

The associate editor coordinating the review of this manuscript and approving it for publication was Shuping He.

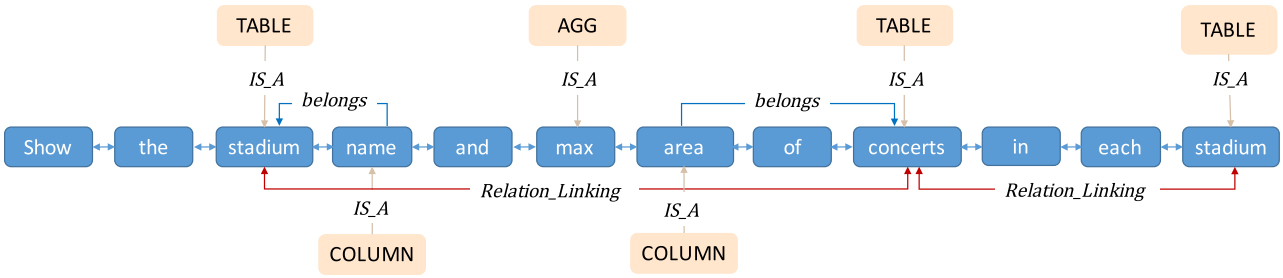

FIGURE 2. The schema graph for the question " Show the stadium name and max area of concerts in each stadium.". The blue edge indicates the word order features of the question. The relation IS_A in the yellow edge represents the schema type features, while the relation belongs and Relation_linking represent the schema relation features.

database schema which contains only one table, or involved the same database at both training and test time. Thus, understanding the database schema structure seems less important. However, about 27% mistakes of SyntaxSQL on Spider are caused by the mismatch of complicated relationship across tables and columns. Consequently, understanding the database schema is important for Text-to-SQL on the new released benchmark. Given a complex database schema that the model has never seen, it is still non-trivial to achieve this goal. We regard this challenge as *schema comprehension problem*. Second, given a cross-domain database schema, predicting column names that are partially or implicitly mentioned in NL is challenging. However, the state-of-the-art neural approach surprisingly still struggles with predicting those columns that appears in NL with exactly the same surface form, which indicates that it is still hard for existing approaches to choose corresponding columns from complex database schema. Thirdly, the complexity of SQL components makes it challenging to generate the correct SQL query. For example, SyntaxSQL only achieve 17% in the IUEN (Except, Union Intersect, Nested SQL) causes. We regard these challenges as *column prediction problem* and *SQL generation problem*, respectively.

Considering the three main challenges above, we first propose a new schema-aware graph, that serves as a hierarchical structure representation, aiming to model the relationships between NL question and database schema. Such insight is partly inspired by the success of using syntactic graph information [10] in semantic parsing tasks. We further employ a relational graph encoder to integrate the structure information of schema graph into NL representation.

For the other two challenges, instead of end-to-end synthesizing a SQL query from a question, we propose a decomposed framework, which decomposes the process of decoding SQL query into three stages. The first column prediction module focuses on predicting a set of candidates from the given database schema and question. Then, the sketch decoder generate a sketch of the SQL query omitting low-level details, e.g., the aggregation, column and table name in our task. Specifically, the sketch guides the generation process, whereas the column prediction module constrains the search space based on the database schema. Then, the detail completion module fills in missing details based on the NL input and the sketch generated from the sketch decoder and the candidates predicted by the column prediction module.

On the challenging Spider [8] benchmark, HSRNet achieves an 45.6% exact matching accuracy, significantly outperforming previous approaches. Further analysis and experimental study demonstrate the effectiveness of HSRNet.

In summary, this paper makes the following three main contributions. (1) We propose a new hierarchical schema representation to facilitate the schema comprehension and model the relationship between question and schema. (2) We introduce a decomposed framework for the Text-to-SQL task, which aims to mitigate the challenge on predicting columns and complex SQL components. (3) We propose a neural approach model for synthesizing SQL, taking NL question and database schema as input. This model significantly outperforms various strong baselines on the Spider benchmark.

II. APPROACH

In this section, we present our approach HSRNet in details. First, we introduce the SQL-specific grammar that we used to guide the decoding process. Then, we describe the hierarchical schema representation and the decomposed framework.

A. METHOD OVERVIEW

Given an NL specification $x = x_1 \cdots x_{|x|}$, our task is to generate an SQL query $y_{sql} = y_{sql_1} \cdots y_{sql_{|y_{sql}|}}$. To take the advantage of the well-defined structure of SQL queries, we design a SQL-specific grammar to help guiding the decoder process and constrain the search space. Inspired by many existing researches on grammar model [11], [12], we design it to be a tree-structured grammar, as shown in Fig. 3. The grammar strictly follow the abstract syntax tree (AST) of the SQL grammar, except for the *From* cause, which will be generated based on the selected columns. To generate the grammar y_g from the SQL query y_{sql}, we first initialize a node *Stmt* and construct its sub-trees according to the grammar and corresponding components on the SQL query. Then for each *Column* node, we attach its detail modules, e.g., the node *agg*, *column* and *table*, where can be identified by the aggregation and column part of the SQL query. Fig. 3 shows an example of the grammar.

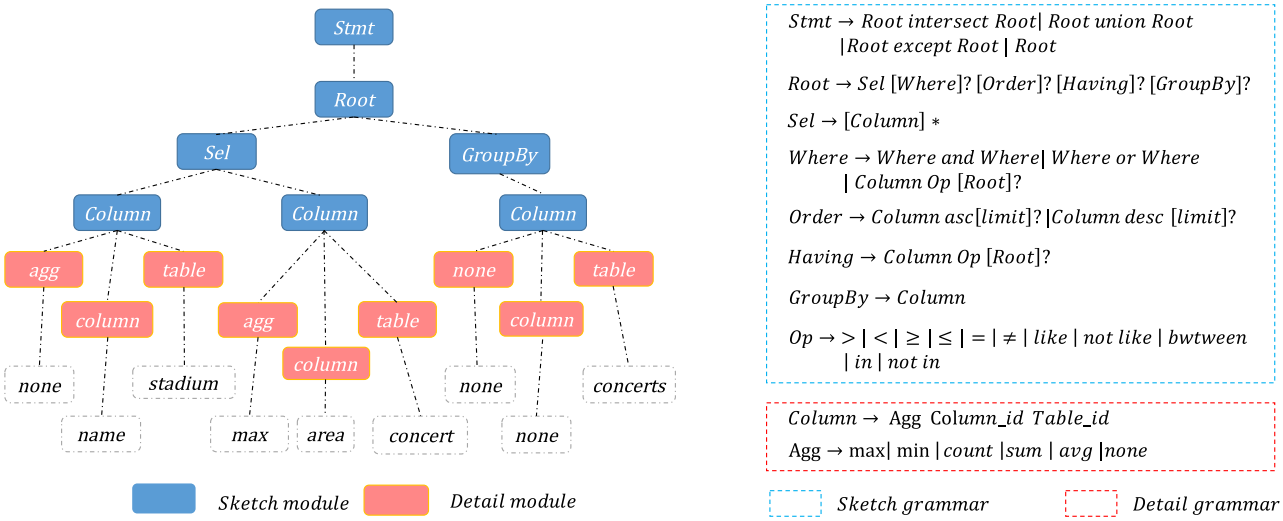

FIGURE 3. **Right The SQL-specific grammar for spider. The cardinality (optional ? and sequential ∗) denotes the number of corresponding symbol. The blue box denotes the sketch part of grammar while the red box denotes the detail part. Left An illustrative example of query " Show the stadium name and max area of concerts in each stadium." and corresponding SQL " SELECT T2.name, max(area) FROM concert AS T1 JOIN stadium AS T2 ON T1.stadium_id = T2.stadium_id GROUP BY T1.stadium_id ".**

The SQL query can be transformed into a sequence of the grammar, which changes the task to generate a sequence of SQL-specific grammar $y_g = y_{g_1} \cdots y_{g_{|y_g|}}$.

B. HIERARCHICAL SCHEMA GRAPH

To represent the relationship between question and schema, we define three types of hierarchical relational features and incorporate them into NL representation. For each input question and schema, we build a schema graph, as shown in Fig. 2. We describe the relational features below.

1) SCHEMA TYPE FEATURES

To enable our model for generalizing unseen databases, the schema type features recognizes the schema type of the word mentioned in NL question. Taking Fig.2 as an example, the word *name* is explicitly annotated as the *"Column"* type, which is critical for our model to predict the corresponding column. As a whole, we define six entity types that may be mentioned in NL question, namely, *Table*, *Column*, *Value*, *Aggregation*, *Comparative*, and *Superlative*, where *Value* stands for a cell value in a database and *Aggregation* represents the aggregation keywords such as Sum, Max and Min.

2) SCHEMA RELATION FEATURES

Besides the type information, the relationship between columns and tables and among the tables with foreign keys should also be taken into account when understanding the schema structure. The relationship between columns and tables help choosing the table when a column is determined, while the relations among tables help inferring the table in the conditional join of *From* cause. Motivated by this observation, we model these relations explicitly and define two specific types. Type *belongs* will be tagged to a column and table if the column is part of the table. Type *Relation Linking* indicates that the tables are connected by private keys.

3) WORD ORDER FEATURES

Followed a previous work [10], we also add the word order features to incorporate the information into word sequence. Each word is connected to its neighboring words. This type of features incorporates the contextual information of NL question.

C. DECOMPOSED FRAMEWORK

Previous studies generate SQL queries directly from NL question and database schema but still achieving an unsatisfactory performance on the Spider benchmark. Through the spider data study, We find that it is challenging to correctly generate SQL components and columns simultaneously. Motivated by this observation, we propose to factorize the generation process. Inspired by the previous work [5], we decompose the generation process into three following stages that can be formulated as:

$$\begin{aligned} p(y_g|x,s) &= p(y_g|x,a,s,c)p(a|x,s,c)p(c|x,s) \\ &= p(y_g|x,a,s,c)p(a|x,s)p(c|x,s), \end{aligned} \tag{1}$$

where a represents the sketch part of grammar y_g. The symbol c denotes the predicted columns from schema s. We assume that the sketch part is independent from the predicted columns c.

Thus our model comprises three modules, a *Column Prediction* module that chooses columns from the given database schema, a *Sketch Decoder* module that generates the sketch grammar of the SQL defined in Figure 3 and a *Detail Completion* module that selects the corresponding aggregation, column and table from candidates based on the sketch grammar predicted in sketch decoder module. The detailed description

of factorizing these generation processes in our model are provided in the Section III.

III. MODEL

In this section, we present our neural approach HSRNet that takes an NL question and a database schema as inputs and outputs a SQL-specific grammar y_g of the corresponding SQL query. Basically, HSRNet follows the encoder-decoder architecture using a recurrent neural network. The input NL question and database schema are first encoded as vectors that are then decoded into the SQL-specific grammar y_g. To address the challenge in schema comprehension problem, we first propose to build a hierarchical graph with database schema and NL question. Then we use a relational graph encoder to extract the structure information of database schema into the NL representation. To mitigate the challenge on predicting column from complex schema and generating complicate SQL components, HSRNet decomposes the SQL decoding process into three modules, e.g., column prediction, sketch decoder and detail completion module. Further details are provided below.

A. ENCODER

The encoder takes an NL specification x and a database schema s as inputs. The encoder first constructs an embedding for each word in x and average the embedding of words in the span level to get the embedding $\boldsymbol{e_x}$. For each column and table name, the encoder constructs its embedding $\boldsymbol{e_t}$ by taking the average embedding of the words constituting the name.

1) SENTENCE ENCODER

To obtain the sentence feature of input x, the encoder runs a bi-directional long short-term memory network [13] (BI-LSTM) over the spans. We concatenate the output hidden states of the forward and backward LSTMs to get the NL question representation $\boldsymbol{L_x}$:

$$\begin{aligned}\overrightarrow{\boldsymbol{L}}_x^t &= \mathrm{f_{LSTM}}\left(\overrightarrow{\boldsymbol{L}}_x^{t-1}, \boldsymbol{e}_x^t\right), \quad t = 1, \cdots, |x| \\ \overleftarrow{\boldsymbol{L}}_x^t &= \mathrm{f_{LSTM}}\left(\overleftarrow{\boldsymbol{L}}_x^{t-1}, \boldsymbol{e}_x^t\right), \quad t = 1, \cdots, |x| \\ \boldsymbol{L_x} &= \left[\overrightarrow{\boldsymbol{L}}_x, \overleftarrow{\boldsymbol{L}}_x\right]. \end{aligned} \tag{2}$$

2) SCHEMA ENCODER

Then, the encoder further considers the structure information of the schema and applies a relational graph encoder to the schema graph $\mathcal{E}_s$ mentioned in Section II-B. Inspired by the successful application of graph convolutional networks (GCN) on knowledge graphs [14], [15], we employ a relational GCN to encode our schema features. Given the nodes and edges $((n_i, e, n_j) \in \mathcal{E}_s)$

$$g_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r} W_r^{(l)} g_j^{(l)} + b^{(l)}\right), \tag{3}$$

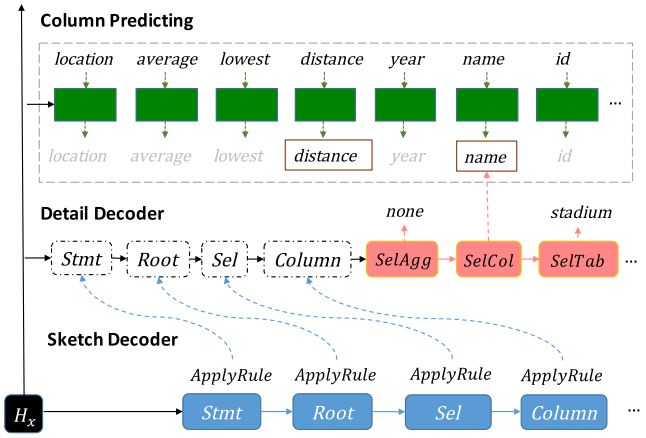

FIGURE 4. The decoder is decomposed into *sketch decoder* (blue box), *column prediction* (green box) and *detail completion* (pink box). The detail completion module fills the sketch of SQL and chooses the columns from column prediction module.

where $g_i^{(l)}$ denotes the hidden state of the node n_i in the layer l, with g_i^0 being initialized by the pre-trained embedding. $\mathcal{R}$ and $\mathcal{N}_i^r$ denote the relation and the set of neighbors under the relation r of node n_i respectively. W and b are trainable parameters, and σ is the activation function.

We denote the output hidden states of the schema encoder as $\boldsymbol{G_x}$. The schema-aware representation $\boldsymbol{H_x}$ of input question x is the concatenation of the schema encoding and word encoding:

$$\boldsymbol{H_x} = [\boldsymbol{G_x}, \boldsymbol{L_x}]. \tag{4}$$

To further address the column prediction problem, we incorporate an attention mechanism to let the representation of columns and tables be aware of the query. Taking the i-th column representation as an example, the enhanced representation $\hat{\boldsymbol{e}}_c^i$ for a column is calculated as follows:

$$\begin{aligned} g_k^i &= \frac{(\boldsymbol{e}_c^i)^{\mathsf{T}} \boldsymbol{e}_x^k}{||\boldsymbol{e}_c^i|| ||\boldsymbol{e}_x^k||} \\ \boldsymbol{c}_c^i &= \sum_{k=1}^{L} g_k^i \boldsymbol{e}_x^k \\ \hat{\boldsymbol{e}}_c^i &= \boldsymbol{e}_c^i + \boldsymbol{c}_c^i, \end{aligned} \tag{5}$$

We denote the enhanced embedding of columns and tables as $\boldsymbol{E}_t$ and $\boldsymbol{E}_c$, respectively.

B. DECODER

To mitigate the challenge on predicting column and predicting SQL components simultaneously, We decompose our decoder into three stages, as shown in Fig. 4. Given the representation vector $\boldsymbol{H_x}$ of NL question and $\boldsymbol{E}_t$ and $\boldsymbol{E}_c$ of schema, the column prediction module learns to choose corresponding columns from complex database schema. The sketch decoder and detail completion modules are a variant of the grammar model [11], which uses the LSTM to model the generation process of a SQL query into sequential applications of actions. We defer the detailed descriptions below.

1) COLUMN PREDICTION

The possibility of the t-th column in the schema is computed by the context vector $\boldsymbol{c}_c^t$ and the enhanced embedding of column $\hat{\boldsymbol{e}}_c^t$:

$$p(col_t|x,s) = sigmoid\left(w_c \cdot \boldsymbol{c}_c^t + w_e \cdot \hat{\boldsymbol{e}_c^t}\right), \tag{6}$$

where w_c and w_c are trainable parameters in our model.

The detail completion module chooses columns from top-k candidates sorted by the possibility $p(col|x, s)$ and the k being the number of columns inferred from the sketch part that predicted by sketch decoder.

2) SKETCH DECODER

The sketch decoder uses the LSTM to model the generation process of the sketch grammar. We define this sequential applications of actions as ApplyRule, which is responsible for applying a production rule to the current derivation tree. Each action step corresponds to a time step in the LSTM. The probability of generating a sketch a is formalized as follows:

$$\begin{aligned} p(a|x,s,c) &= p(a|x,s) \\ &= \prod_{i=1}^{T} p(a_i = \text{ApplyRule}[r]|x,s,a_{<i}), \end{aligned} \tag{7}$$

where a_i denotes an action taken at time step i and $a_{<i}$ is the sequence of actions before step i. The ApplyRule$[r]$ stands the decoder chooses the r-th ApplyRule action in the step i. The probability of selecting a rule r is calculated as follows:

$$\begin{aligned} \boldsymbol{h}_i &= \text{LSTM}([\boldsymbol{a}_{i-1}; \boldsymbol{n}_{i-1}; \boldsymbol{v}_{i-1}], \boldsymbol{h}_{i-1}) \\ \boldsymbol{w}_i &= \text{softmax}(\boldsymbol{H}_x \boldsymbol{W}_a \boldsymbol{h}_i) \\ \boldsymbol{c}_i &= \boldsymbol{w}_i^{\top} \boldsymbol{H}_x \\ \boldsymbol{v}_i &= \tanh(\boldsymbol{W}_v[\boldsymbol{h}_i; \boldsymbol{c}_i] + \boldsymbol{b}_v) \\ p(a_i &= \text{ApplyRule}[r]|x,s,a_{<i}) \propto \\ &\quad \exp(\boldsymbol{W}_r(\tanh(\boldsymbol{W}_p \boldsymbol{v}_i + \boldsymbol{b}_p))), \end{aligned} \tag{8}$$

where $\boldsymbol{a}_{i-1}$ is the action embedding of previous step, $\boldsymbol{n}_{i-1}$ is the embedding of the type of frontier node and $\boldsymbol{h}_i$ is the hidden states of LSTM. $\boldsymbol{h}_0$ is the final hidden states of the encoder BI-LSTM. $\boldsymbol{W}_a$, $\boldsymbol{W}_v$, $\boldsymbol{W}_p$, $\boldsymbol{W}_r$, $\boldsymbol{b}_v$ and $\boldsymbol{b}_p$ are trainable parameters. We assign each product rule an embedding. When the action is ApplyRule, we take the embedding of the production rule as the action embedding.

3) DETAIL COMPLETION

Similar to sketch decoder, the detail completion module uses the LSTM to generate the sequential actions of selecting aggregation, columns or tables to fill in the missing details in the sketch. Thus we define three types of actions, i.e., SelAgg, SelectCol and SetTab to represent the generation:

$$\begin{aligned} p(y_g|x,a,s,c) = \prod_{i=1}^{T_c} [&\lambda_i^c p(a_i = \text{SelCol}[c]|x,s,c,a_{<i}) \\ &+ (\lambda_i^t) p(a_i = \text{SelTab}[t]|x,s,c,a_{<i}) \\ &+ (\lambda_i^a) p(a_i = \text{SelAgg}[t]|x,s,c,a_{<i})], \end{aligned} \tag{9}$$

where $\lambda_i \in \{\lambda_i^c, \lambda_i^t, \lambda_i^a\}$ is 1 when the i-th action is corresponding type, otherwise $\lambda_i = 0$. The symbol c denotes the column set predicted by the column prediction module. We leverage pointer network [16] to implement the SelectAgg, SelectCol and SelectTab function.

C. TRAINING AND INFERENCE

First we minimize the following log-likelihood loss of column prediction module:

$$\begin{aligned} loss(col, x, s) = -\sum_{j=1}^{C} &\left(y_j \log p\left(col_j|x,s\right)\right. \\ &\left. + \left(1 - y_j\right) \log\left(1 - p\left(col_j|x,s\right)\right)\right), \end{aligned} \tag{10}$$

where y_j indicates the ground true label of the j-th column.

The model is optimized by the multi-task of maximizing the log-likelihood of the ground truth action sequences and minimizing the loss of column prediction module:

$$\begin{aligned} max \sum_{(x,s,y_g)\in\mathcal{D}} &\left(\log p(y_g|a,x,s) + \lambda \log p(a|x,s)\right) \\ &-\alpha loss(col,x,s), \end{aligned} \tag{11}$$

where $\mathcal{D}$ represents training pairs of SQL queries. λ and α represents the scale between $\log p(y_g|a,x,s)$ and $\log p(a|x,s)$, which all set to 1 in our experiment.

At inference time, we first obtain a sketch a^* via argmax $p(a|x, s)$ and column set c predicted from the database schema. Then, generate a sequence of SQL-grammar y_g^* via argmax $p(y_g|x, s, c, a)$ from the sketch and column set predicted above. Instead of iterating over all candidates, we use greedy search to approximate the best result.

IV. EXPERIMENT

In this section, we evaluate the effectiveness of our model by comparing HSRNet to various baseline approaches. We also conduct a thorough ablation study and an error analysis on our model. Code for HSRNet is available at https://github.com/NoviceCookie/HSRNet

A. EXPERIMENT SETUP

1) DATASET

We conduct our experiments on Spider [8], a new large-scale human-annotated and cross-domain Text-to-SQL dataset with complex SQL queries. We use the database split, as mentioned in previous setting [9], where 206 databases are split into 146 training, 20 development and 40 test. In total, there are 8625, 1034, 2000 examples for training, development and test. The SQL queries are divided into 4 hardness levels: easy, medium, hard and extra hard based on the number of SQL components. We pre-process examples to remove stop words and lemma words in the input question. We evaluate HSRNet and other approaches using the SQL Exact Matching and Component Matching proposed by Spider [8]. The test set of Spider is unreleased, and our model are submitted to the owner once for getting official scores on the hidden test set.

TABLE 1. Accuracy of exact matching score on SQL queries of HRSNet and other baselines on development set and test set.

Approach	Dev	Test				
	All	Easy	Medium	Hard	Extra Hard	All
Seq2Seq	1.9%	11.9%	1.9%	13.0%	0.5 %	3.7%
Seq2Seq + Attention	1.8%	14.9%	2.5%	2.0%	1.1%	4.8%
Seq2Seq + Copying	4.1%	15.4%	3.4%	2.0%	1.1%	5.3%
TypeSQL	8.0%	19.6%	7.6%	3.8%	0.8%	8.2%
SQLNet	10.9%	26.2%	12.6%	6.6%	1.3%	12.4%
SyntaxSQLNet	18.9%	38.6%	17.6%	16.3%	4.9%	19.7%
HSRNet	**51.5%**	**70.1%**	**48.8%**	**38.4%**	**15.9%**	**45.6%**

2) BASELINES

To establish the baseline, we also evaluate a basic sequence-to-sequence model [17] and explore it augmented with neural attention mechanism [18] and copying mechanism [19]. We also include SQLNet [2] which employs a sketch-based method and synthesize SQL as a slot-filling task based on the sketch, and TypeSQL [4] improved upon SQLNet by utilizing types information. In addition, we explore the prior state-of-the-art method SyntaxSQLNet [9], which employs a SQL specific syntax tree-based decoder with SQL generation path history and table-aware column attention encoders.

3) IMPLEMENTATION DETAILS

We implement HSRNet and the baseline approaches in PyTorch [20]. Dimensions of word embedding and hidden vectors are set to 300. Word embedding are initialized by Glove [21], shared by sentence tokens and tokens in database schema, and are fixed during training. The dimensions of schema encoder is set to 300 and initialized by the Glove embedding and the depth of our layers is set to 2. The dimension of action embedding and node type embedding are set to 128 and 64 respectively. The dropout rate is 0.3. We use Adam [22] with default hyperparameters for optimization and the training loss scale λ and α is set to 1 respectively. Batch size is set to 64.

B. EXPERIMENT RESULT

1) COMPARISON WITH BASELINE

Table 1 presents the accuracy of exact matching score of HSRNet and various baselines on the development set and test set. As can be noticed from Table 1, HSRNet clearly outperforms various baselines. It achieves 45.6% exact matching score on all SQL queries, and registers 25.9% absolute improvements over SyntaxSQLNET in accuracy.

2) EXPERIMENT ON SCHEMA ENCODING

To better demonstrate the effectiveness of encoding structure information into neural model, we alter the baseline approaches with the schema encoder. As shown in Table 2, there are at least 6.2% and up to 8.6% absolute improvements on accuracy of exact matching score on development set. We conjecture that the type information of TypeSQL overlaps the schema type features of hierarchical schema graph so the improvement is not significant compared to other approaches.

We further explore the effectiveness of type and relation features. Fig. 5 and Fig. 6 show the cosine similarity of

TABLE 2. Accuracy of exact matching on development set. The header 'SQL' means that the approach are learned to generate SQL, which the header 'SemQL' indicates that they are learned to generate SQL query.

Approach	Base	+SE	Diff.
Seq2Seq	1.9%	10.5%	+8.6%
TypeSQL	8.0%	14.2%	+6.2%
SQLNet	10.9%	18.1%	+7.2%
SyntaxSQLNet	18.9%	26.6%	+7.7%

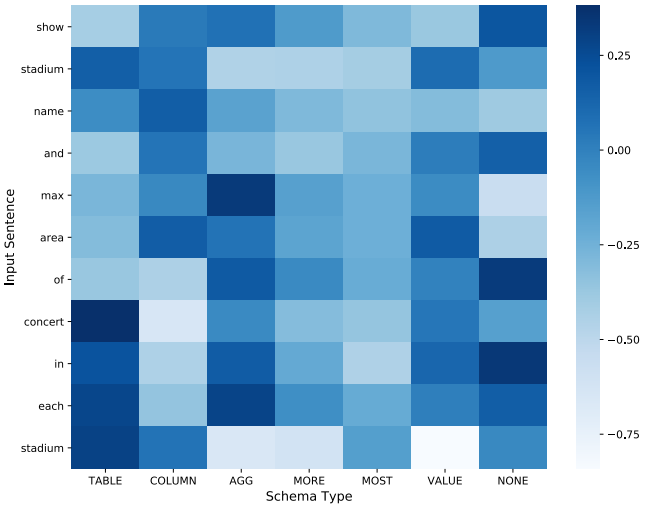

FIGURE 5. The cosine similarity of schema encoder output vector G_x and the embedding of entity types of schema type features.

schema encoder output vector $\boldsymbol{G_x}$ and the embedding of corresponding types. After incorporating the schema type features, the output vector $\boldsymbol{G_x}$ captures the alignment of schema types, as shown in Fig. 5. For example, the output vector of schema encoder of word 'concert' is most similar to the type *Table*. Fig. 6 demonstrates the effectiveness of the schema relation features. The column 'name' is belong to the table 'stadium' and the output vector $\boldsymbol{G_{name}}$ is similar to the embedding of table 'stadium'.

Fig. 7 presents the accuracy of exact matching score of using different schema features by the hardness level. It is clear that using on all combinations of features is significantly outperforms the base model in all four hardness levels, and the gain of the schema encoding mainly comes from the relation features and type features. The relation features empower the model on predicting complex schema database and improve the result on extra hard level. The type features help model better understand corresponding entity types in the question.

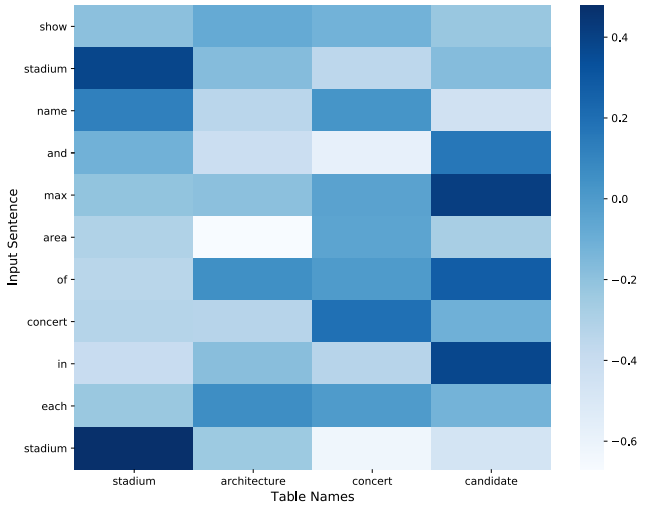

FIGURE 6. The cosine similarity of schema encoder output vector G_x and the embedding of table names.

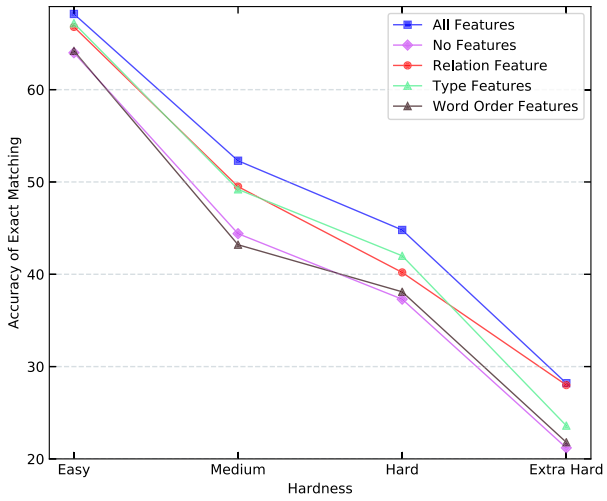

FIGURE 7. Accuracy of exact matching score of different schema features on development set by hardness level.

The SQL query that belongs to Hard and Extra Hard level is much more complicated in terms of the structure of database schema, and the gain from these harness indicates that the schema graph representation mainly addresses the schema comprehension problem.

Fig. 8 presents the exact matching accuracy of model with and without schema encoding as a function of the number of tables in the SQL query. As shown in figure, the performance increases when the database has more mentioned tables. The model without schema encoding lacks of the comprehension on database schema while the model incorporated with schema features performs better than base model. The performance of with and without schema encoding both degrade to nearly 10% when the number of tables comes to 4. One cause may be the difficulty of choosing correct tables in the *From* cause from many candidates in a complex database schema, which is still changeling for the current approach.

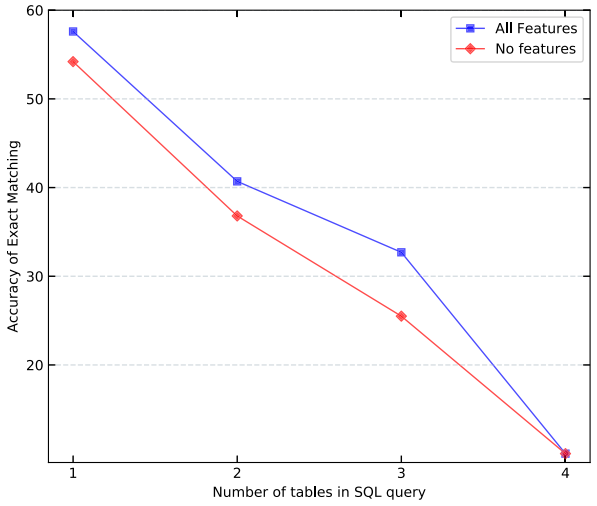

FIGURE 8. Accuracy of exact matching score as a function of the number of tables in SQL query.

TABLE 3. F1 scores of component matching of ablation study. Base model means that we does not perform schema encoder(SE) and decompose framework on column prediction(CP) and sketch generation(SeG).

Technique	Dev				
	Select	*Where*	*GroupBy*	*OrderBy*	All
Base model	73.7%	42.8%	64.3%	65.6%	39.7%
+SE	77.9%	51.2%	66.5%	69.8%	47.4%
+SE+SeG	78.9%	51.9%	**69.6%**	**72.4%**	49.8%
+SE+SeG+CP	**80.4%**	**55.9%**	68.6%	71.1%	**51.5%**

C. ABLATION STUDY

To further understand the effectiveness of HSRNet, we ablate our approach to analyze the contribution of each technique on development set.

"+SE" adds the schema encoder into the model, which focuses on modeling the database schema structure information. The performance improves significantly when incorporating the information of the database schema structure. The gain mainly comes from the *Where* cause, which needs the schema information to infer the column and the components of subquery. The schema type features assigns words on question with its corresponding type, which could help model identify some keywords on SQL quires and improve the performance on the *Select* cause. "+CP" and "+SeG" adds the column prediction and sketch generation module respectively. As Table. 3 shows, removing the sketch decoder harms the performance since the decoder loses access to additional high-level information of the generated sequence. The column prediction module, on the other hands, mitigates the difficulty of choosing right columns from the complex database schema. The result of our ablation study demonstrates the effectiveness of the proposed Text-to-SQL model.

D. ERROR ANALYSIS

To understand the sources of errors, we analyze 502 cases which are failed in exact matching synthesized by our model from development set. 47.3% of these errors are due to the mismatch of column and most of them are caused by

the conditional value. Although the type relation features of schema graph can alleviate the column matching problem to some extends, it is still hard for model to understand the conditional value without access to database content. A good representative example is 'Count the number of flights into ATO', wherein the conditional value 'ATO' is belong to column 'SourceAirport'. To address the cases where only the value of a column is mentioned, as the study points out [23], the content of a database is required.

23.9% of them fail to predict correct SQL sketch, which lead to an accuracy lower than 40% in the extra hard level. There are still 21 cases failed in predicting self-join in the *From* cause. Besides, the inconsistent of *GroupBy* label also causes 15 cases failed. There are some other errors remained in SQL diversity and label error cases.

V. RELATED WORK

Since 1970s, the task of Text-to-SQL or NLIDB (Natural Language Interface to Database) has received a wide range of significant attention [24]–[27]. The early methods proposed in the community are hand-crafted to a specific database and tend to involve hand features [24], [28], [29], making it challenging to generalize to new domains. These systems has shown within a limited subset of English can be translated into a certain subset of logical forms, such as programming language and SQL query. These works establish a good foundation of NLIDB. To reuse the NLIDB systems for multiple databases, later works focus on the generalization and effectiveness of these systems [30]–[32]. These systems are adapted to interface with new databases by users who are not experts in natural-language processing. However, the proposed rule-based systems still have poor performance on unseen query and new database, which require more semantic understanding of input question.

Recent years have seen a great deal of interest in neural network-based methods to address the semantic parsing task [2], [4], [6], [7], [9], [33]–[35]. Among them, sketch-based methods and generated-based methods have been the two most successful approaches. The sketch-based approaches predict the content for each slot based on the pre-designed sketch so that it changes the generation to slot-filling task. The generated-based methods can generate nested and complex queries because of the general decoding process. But the complex of enormous search space prohibits the generated-based methods achieving satisfactory performance. With the release of large-scale Text-to-SQL benchmarks such as WikiSQL [1] and Spider [8], there is a renewed interest in the NLIDB. State-of-the-art approaches on WikiSQL have achieved accuracy beyond 85%. However, the best results on Spider benchmark which targets a more real-world setting, are still far from being satisfactory. Inspired by a recent study on WikiSQL about the challenges towards 100% condition accuracy [23], we conduct an analysis on Spider benchmark and design our approach for the complex and cross-domain Text-to-SQL task.

Inspired by recent studies on Graph-to-Sequence model [10], [42], we focus on the information between question and database schema and further design a hierarchical schema representation. Unlike previous works on graph representation which focus on modeling the syntactic information in semantic parsing, our approach aims to incorporate the schema information into the text representation. Our schema encoder is closely related to a number of works in the area of neural networks on graphs. It is primarily motivated as an adaption of previous works on graph convolutional networks (GCN) [39]–[41]. Similarly to the RGCN [14], we employ a Relational graph encoder to model the graph structure with different relationship types.

Another lines of research that are related to our work is the structured neural decoders [11], [36]–[38]. The decoder of HSRNet is based on the grammar model [11]. We extend the grammar model to accommodate the characteristics of SQL query.

VI. CONCLUSION

In this paper, we propose a novel neural semantic parsing model called HSRNet for Text-to-SQL task and validate its effectiveness on the complex and cross-domain benchmark, Spider. HSRNet encodes the structure of the database schema with graph convolution network and incorporate the information into sentence representation. This additional schema features have improved at least 6.2% and up to 8.6% absolute improvements of our baselines on the development set of Spider. We also carefully conduct experiments to demonstrate the effect of each features in the hierarchical schema graph. To address the challenging on predicting columns from complex schema and generating complicate SQL quires, we propose a decoder framework, which decomposes the process into column prediction, sketch generation and detail completion. Experiment shows the effectiveness of our methods.

In the future, once more Text-to-SQL datasets are released, we will validate our model and conduct experiments on these datasets. Beside, we will seek to more efficient methods to encode our hierarchical schema representation.

REFERENCES

[1] V. Zhong, C. Xiong, and R. Socher, "Seq2SQL: Generating structured queries from natural language using reinforcement learning," *CoRR*, vol. abs/1709.00103, pp. 1–12, Aug. 2017.

[2] X. Xu, C. Liu, and D. Song, "SQLNet: Generating structured queries from natural language without reinforcement learning," Nov. 2017, *arXiv:1711.04436*. [Online]. Available: https://arxiv.org/abs/1711.04436

[3] N. Yaghmazadeh, Y. Wang, I. Dillig, and T. Dillig, "SQLizer: Query synthesis from natural language," *Proc. ACM Program. Lang.*, vol. 1, pp. 63:1–63:26, Oct. 2017. doi: 10.1145/3133887.

[4] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. Radev, "TypeSQL: Knowledge-based type-aware neural text-to-SQL generation," in *Proc. 16th Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, vol. 2. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2018, pp. 588–594. [Online]. Available: http://aclweb.org/anthology/N18-2093

[5] L. Dong and M. Lapata, "Coarse-to-fine decoding for neural semantic parsing," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2018, pp. 731–742. [Online]. Available: http://aclweb.org/anthology/P18-1068

[6] C. Wang, K. Tatwawadi, M. Brockschmidt, P.-S. Huang, Y. Mao, O. Polozov, and R. Singh, "Robust text-to-SQL generation with execution-guided decoding," Jul. 2018, *arXiv:1807.03100*. [Online]. Available: https://arxiv.org/abs/1807.03100

[7] W. Hwang, J. Yim, S. Park, and M. Seo, "A comprehensive exploration on WikiSQL with table-aware word contextualization," Feb. 2019, *arXiv:1902.01069*. [Online]. Available: https://arxiv.org/abs/1902.01069

[8] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2018, pp. 3911–3921. [Online]. Available: http://aclweb.org/anthology/D18-1425

[9] T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li, and D. Radev, "SyntaxSQLNet: Syntax tree networks for complex and cross-domaintext-to-SQL task," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2018, pp. 1653–1663. [Online]. Available: http://aclweb.org/anthology/D18-1193

[10] K. Xu, L. Wu, Z. Wang, M. Yu, L. Chen, and V. Sheinin, "Exploiting rich syntactic information for semantic parsing with graph-to-sequence model," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Brussels, Belgium: Assoc. Comput. Linguistics, Oct./Nov. 2018, pp. 918–924. [Online]. Available: https://www.aclweb.org/anthology/D18-1110

[11] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2017, pp. 440–450. [Online]. Available: http://aclweb.org/anthology/P17-1041

[12] P. Yin and G. Neubig, "TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2018, pp. 7–12. [Online]. Available: http://aclweb.org/anthology/D18-2002

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Proc. ESWC*, 2018, pp. 593–607.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, pp. 1–14, Sep. 2016.

[16] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2015, pp. 2692–2700. [Online]. Available: http://papers.nips.cc/paper/5866-pointer-networks.pdf

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 2. Cambridge, MA, USA: MIT Press, 2014, pp. 3104–3112. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969173

[18] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," Sep. 2014, *arXiv:1409.0473*. [Online]. Available: https://arxiv.org/abs/1409.0473

[19] J. Gu, Z. Lu, H. Li, and V. O. Li, "Incorporating copying mechanism in sequence-to-sequence learning," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2016, pp. 1631–1640. [Online]. Available: http://aclweb.org/anthology/P16-1154

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *Proc. NIPS-W*, 2017, pp. 1–4.

[21] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2014, pp. 1532–1543. [Online]. Available: http://aclweb.org/anthology/D14-1162

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980

[23] S. Yavuz, I. Gur, Y. Su, and X. Yan, "What it takes to achieve 100% condition accuracy on WikiSQL," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2018, pp. 1702–1711. [Online]. Available: http://aclweb.org/anthology/D18-1197

[24] D. H. D. Warren and F. C. N. Pereira, "An efficient easily adaptable system for interpreting natural language queries," *Comput. Linguistics*, vol. 8, nos. 3–4, pp. 110–122, Jul./Dec. 1982. [Online]. Available: http://dl.acm.org/citation.cfm?id=972942.972944

[25] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch, "Natural language interfaces to databases—An introduction," *Natural Lang. Eng.*, vol. 1, no. 1, pp. 29–81, 1995.

[26] A.-M. Popescu, A. Armanasu, O. Etzioni, D. Ko, and A. Yates, "Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability," in *Proc. 20th Int. Conf. Comput. Linguistics (COLING)*. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2004, p. 141. doi: 10.3115/1220355.1220376.

[27] C. Hallett, "Generic querying of relational databases using natural language generation techniques," in *Proc. 4th Int. Natural Lang. Gener. Conf. (INLG)*. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2006, pp. 95–102. [Online]. Available: http://dl.acm.org/citation.cfm?id=1706269.1706289

[28] W. A. Woods, "Semantics and quantification in natural language question answering," in *Readings in Natural Language Processing*. B. J. Grosz, K. Sparck-Jones, and B. L. Webber, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1986, pp. 205–248. [Online]. Available: http://dl.acm.org/citation.cfm?id=21922.24336

[29] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a natural language interface to complex data," *ACM Trans. Database Syst.*, vol. 3, no. 2, pp. 105–147, Jun. 1978. doi: 10.1145/320251.320253.

[30] B. J. Grosz, D. E. Appelt, P. A. Martin, and F. C. N. Pereira, "Team: An experiment in the design of transportable natural-language interfaces," *Artif. Intell.*, vol. 32, no. 2, pp. 173–243, May 1987. doi: 10.1016/0004-3702(87)90011-7.

[31] I. Androutsopoulos, G. Ritchie, and P. Thanisch, "Masque/SQL: An efficient and portable natural language query interface for relational databases," in *Proc. 6th Int. Conf. Ind. Eng. Appl. Artif. Intell. Expert Syst. (IEA/AIE)*. Philadelphia, PA, USA: Gordon Breach Sci. Publishers, 1993, pp. 327–330. [Online]. Available: http://dl.acm.org/citation.cfm?id=1114022.1114073

[32] L. R. Tang and R. J. Mooney, "Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing," in *Proc. Joint SIGDAT Conf. Empirical Methods Natural Lang. Process. Very Large Corpora, Held Conjunction 38th Annu. Meeting Assoc. Comput. Linguistics (EMNLP)*, vol. 13. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2000, pp. 133–141. doi: 10.3115/1117794.1117811.

[33] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer, "Learning a neural semantic parser from user feedback," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2017, pp. 963–973. [Online]. Available: http://aclweb.org/anthology/P17-1089

[34] Y. Sun, D. Tang, N. Duan, J. Ji, G. Cao, X. Feng, B. Qin, T. Liu, and M. Zhou, "Semantic parsing with syntax-and table-aware SQL generation," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2018, pp. 361–372. [Online]. Available: http://aclweb.org/anthology/P18-1034

[35] I. Gur, S. Yavuz, Y. Su, and X. Yan, "DialSQL: Dialogue based structured query generation," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2018, pp. 1339–1349. [Online]. Available: http://aclweb.org/anthology/P18-1124

[36] L. Dong and M. Lapata, "Language to logical form with neural attention," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2016, pp. 33–43. [Online]. Available: http://aclweb.org/anthology/P16-1004

[37] C. Xiao, M. Dymetman, and C. Gardent, "Sequence-based structured prediction for semantic parsing," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1. Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2016, pp. 1341–1350. [Online]. Available: http://aclweb.org/anthology/P16-1127

[38] J. Krishnamurthy, P. Dasigi, and M. Gardner, "Neural semantic parsing with type constraints for semi-structured tables," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Stroudsburg, PA, USA: Assoc. Comput. Linguistics, 2017, pp. 1516–1526. [Online]. Available: http://aclweb.org/anthology/D17-1160

[39] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2016, pp. 3844–3852. [Online]. Available: http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering.pdf

[40] D. K. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2015, pp. 2224–2232. [Online]. Available: http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints.pdf

[41] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2014, pp. 1–14.

[42] B. Bogin, M. Gardner, and J. Berant, "Representing schema structure with graph neural networks for text-to-SQL parsing," May 2019, *arXiv:1905.06241*. [Online]. Available: https://arxiv.org/abs/1905.06241

MEINA SONG received the Ph.D. degree in electronic engineering from the Beijing University of Posts and Telecommunications, in 2004, where she is currently a Professor with the School of Computer Science. She has published hundreds of academic or technical papers on journals and conferences and served as the Director of the Engineering Research Center on Information Network for Education Ministry. Her research interests include the mobile Internet, cloud computing, big data, and artificial intelligence. She also served as a Technical Committee Member on data communications and service computing for China Computer Federation.

ZECHENG ZHAN received the B.S. degree from the School of Optical Information Science and Technology, Beijing Institute of Technology, Beijing, China, in 2013. He is currently pursuing the M.S. degree with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing. He has published several papers and patents in related fields, and there are some excellent open source projects on his GitHub, such as Semantic Parsing. His research interests include natural language processing and recommendation algorithm.

HAIHONG E. received the B.S. degree from the School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2005, and the M.S. and Ph.D. degrees from the School of Computer Science, Beijing University of Posts and Telecommunications, in 2010, where she is currently an Associate Professor.

From 2010 to 2014, she was an Associate Team Leader with the China Communications Standards Association TC11-WG3. Since 2014, she has been an Associate Secretary General Ministry of Science and Technology Modern Service Industry Common Service Alliance. Her research interest includes the big data and artificial intelligence, mainly on network representation learning and graph-based data mining.

Dr. E's awards and honors include the Key Technologies and Applications of Common Service Platforms in Modern Service Industry Science and Technology Progress Award of Higher Education Scientific Research Outstanding Award Second Prize China Ministry of Education, the Third Prize of 10 Industry Standards for Overall Technical Requirements for Mobile User Personal Information Management Business by China Communications Standards Association Science and Technology Progress Award, Common Service Platform for Modern Service Industry China Service Industry Technology Innovation Award Special Award China Business Federation.