Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing

Ben Bogin[1] **Matt Gardner**[2] **Jonathan Berant**[1,2]

[1]School of Computer Science, Tel-Aviv University
[2]Allen Institute for Artificial Intelligence

ben.bogin@cs.tau.ac.il,mattg@allenai.org,joberant@cs.tau.ac.il

Abstract

Research on parsing language to SQL has largely ignored the structure of the database (DB) schema, either because the DB was very simple, or because it was observed at both training and test time. In SPIDER, a recently-released text-to-SQL dataset, new and complex DBs are given at test time, and so the structure of the DB schema can inform the predicted SQL query. In this paper, we present an encoder-decoder semantic parser, where the structure of the DB schema is encoded with a graph neural network, and this representation is later used at both encoding and decoding time. Evaluation shows that encoding the schema structure improves our parser accuracy from 33.8% to 39.4%, dramatically above the current state of the art, which is at 19.7%.

1 Introduction

Semantic parsing (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005) has recently taken increased interest in parsing questions into SQL queries, due to the popularity of SQL as a query language for relational databases (DBs).

Work on parsing to SQL (Zhong et al., 2017; Iyer et al., 2017; Finegan-Dollak et al., 2018; Yu et al., 2018a) has either involved simple DBs that contain just one table, or had a single DB that is observed at both training and test time. Consequently, modeling the *schema structure* received little attention. Recently, Yu et al. (2018b) presented SPIDER, a text-to-SQL dataset, where at test time questions are executed against unseen and complex DBs. In this zero-shot setup, an informative representation of the schema structure is important. Consider the questions in Figure 1: while their language structure is similar, in the first query a 'join' operation is necessary because the information is distributed across three tables, while in the other query no 'join' is needed.

x : *Find the age of students who **do not have** a cat pet.*
y : `SELECT age FROM student WHERE student NOT IN (SELECT ... FROM student **JOIN** has_pet ... **JOIN** pets ... WHERE ...)`
x : *What are the names of teams that **do not have** match season record?*
y : `SELECT name FROM team WHERE team_id NOT IN (SELECT team FROM match_season)`

Figure 1: Examples from SPIDER showing how similar questions can have different SQL queries, conditioned on the schema. Table names are underlined.

In this work, we propose a semantic parser that strongly uses the schema structure. We represent the structure of the schema as a graph, and use graph neural networks (GNNs) to provide a global representation for each node (Li et al., 2016; De Cao et al., 2019; Sorokin and Gurevych, 2018). We incorporate our schema representation into the encoder-decoder parser of Krishnamurthy et al. (2017), which was designed to parse questions into queries against unseen semi-structured tables. At encoding time we enrich each question word with a representation of the subgraph it is related to, and at decoding time we emit symbols from the schema that are related through the graph to previously decoded symbols.

We evaluate our parser on SPIDER, and show that encoding the schema structure improves accuracy from 33.8% to 39.4% (and from 14.6% to 26.8% on questions that involve multiple tables), well beyond 19.7%, the current state-of-the-art. We make our code publicly available at https://github.com/benbogin/spider-schema-gnn.

2 Problem Setup

We are given a training set $\{(x^{(k)}, y^{(k)}, S^{(k)})\}_{k=1}^{N}$, where $x^{(k)}$ is a natural language question, $y^{(k)}$ is its translation to a SQL query, and $S^{(k)}$ is the schema of the DB where $y^{(k)}$ is executed. Our

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4560–4565
Florence, Italy, July 28 - August 2, 2019. ©2019 Association for Computational Linguistics

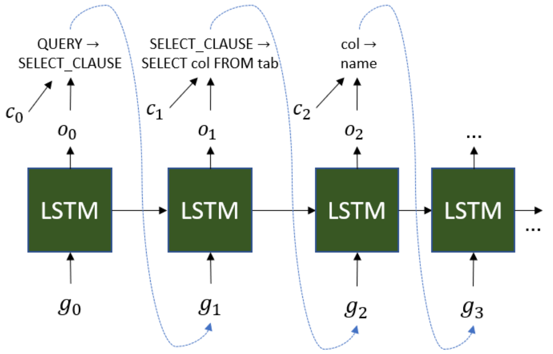

Figure 2: The decoder we base our work on (Krishnamurthy et al., 2017). The input to the LSTM (g_j) at step j is a learned embedding of the last decoded grammar rule, except when the last rule is schema-specific (g_3), where the input is a learned embedding of the schema item type. A grammar rule is selected based on the LSTM output (o_j) and the attended hidden state of the input LSTM (c_j).

goal is to learn a function that maps an unseen question-schema pair (x, S) to its correct SQL query. Importantly, the schema S was not seen at training time, that is, $S \neq S^{(k)}$ for all k.

A DB schema S includes: (a) The set of DB tables $\mathcal{T}$ (e.g., `singer`), (b) a set of columns $\mathcal{C}_t$ for each $t \in \mathcal{T}$ (e.g., `singer_name`), and (c) a set of foreign key-primary key column pairs $\mathcal{F}$, where each $(c_f, c_p) \in \mathcal{F}$ is a relation from a foreign-key c_f in one table to a primary-key c_p in another. We term all schema tables and columns as *schema items* and denote them by $\mathcal{V} = \mathcal{T} \cup \{\mathcal{C}_t\}_{t \in \mathcal{T}}$.

3 A Neural Semantic Parser for SQL

We base our model on the parser of Krishnamurthy et al. (2017), along with a grammar for SQL provided by AllenNLP (Gardner et al., 2018; Lin et al., 2019), which covers 98.3% of the examples in SPIDER. This parser uses a linking mechanism for handling unobserved DB constants at test time. We review this model in the context of text-to-SQL parsing, focusing on components we expand upon in §4.

Linking schema items To handle unseen schema items, Krishnamurthy et al. (2017) learn a similarity score $s_{\text{link}}(v, x_i)$ between a word x_i and a schema item v that has type τ.[1] The score is based on learned word embeddings and a few manually-crafted features.

The linking score is used to compute

$$p_{\text{link}}(v \mid x_i) = \frac{\exp(s_{\text{link}}(v, x_i))}{\sum_{v' \in \mathcal{V}_\tau \cup \{\varnothing\}} \exp(s_{\text{link}}(v', x_i))},$$

where $\mathcal{V}_\tau$ are all schema items of type τ and $s_{\text{link}}(\varnothing, \cdot) = 0$ for words that do not link to any schema item. The functions $p_{\text{link}}(\cdot)$ and $s_{\text{link}}(\cdot)$ will be used to decode unseen schema items.

Encoder A Bidirectional LSTM (Hochreiter and Schmidhuber, 1997) provides a contextualized representation h_i for each question word x_i. Importantly, the encoder input at time step i is $[w_{x_i}; l_i]$: the concatenation of the word embedding for x_i and $l_i = \sum_\tau \sum_{v \in \mathcal{V}_\tau} p_{\text{link}}(v \mid x_i) \cdot r_v$, where r_v is a learned embedding for the schema item v, based on the *type* of v and its schema neighbors. Thus, $p_{\text{link}}(v \mid x_i)$ augments every word x_i with information on the schema items it should link to.

Decoder We use a grammar-based (Xiao et al., 2016; Cheng et al., 2017; Yin and Neubig, 2017; Rabinovich et al., 2017) LSTM decoder with attention on the input question (Figure 2). At each decoding step, a non-terminal of type τ is expanded using one of the grammar rules. Rules are either schema-independent and generate non-terminals or SQL keywords, or schema-specific and generate schema items.

At each decoding step j, the decoding LSTM takes a vector g_j as input, which is an embedding of the grammar rule decoded in the previous step, and outputs a vector o_j. If this rule is schema-independent, g_j is a learned global embedding. If it is schema-specific, i.e., a schema item v was generated, g_j is a learned embedding $\tau(v)$ of its type. An attention distribution a_j over the input words is computed in a standard manner (Bahdanau et al., 2015), where the attention score for every word is $h_i^\top o_j$. It is then used to compute the weighted average of the input $c_j = \sum_i a_j h_j$. Now a distribution over grammar rules is computed by:

$$\begin{aligned} s_j^{\text{glob}} &= \text{FF}([o_j; c_j]) \quad \in \mathbb{R}^{G_{\text{legal}}}, \\ s_j^{\text{loc}} &= S_{\text{link}} a_j \quad \in \mathbb{R}^{\mathcal{V}_{\text{legal}}}, \\ p_j &= \text{softmax}([s_j^{\text{glob}}; s_j^{\text{loc}}]), \end{aligned}$$

where G_{legal}, $\mathcal{V}_{\text{legal}}$ are the number of legal rules (according to the grammar) that can be chosen at time step j for schema-independent and schema-specific rules respectively. The score s_j^{glob} is computed with a feed-forward network, and the score

[1] Types are tables, string columns, number columns, etc.

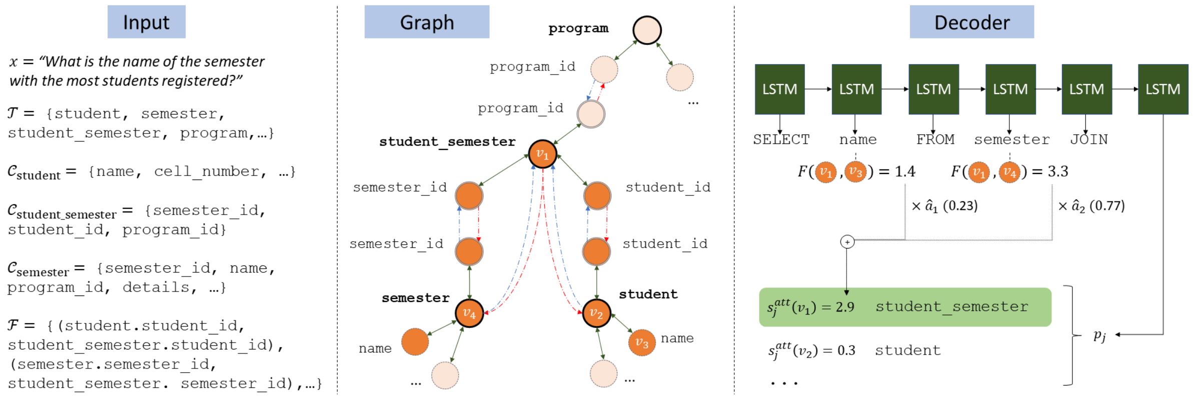
Input
x = "What is the name of the semester with the most students registered?"
𝒯 = {student, semester, student_semester, program,…}
𝒞student = {name, cell_number, …}
𝒞student_semester = {semester_id, student_id, program_id}
𝒞semester = {semester_id, name, program_id, details, …}
ℱ = {(student.student_id, student_semester.student_id), (semester.semester_id, student_semester. semester_id),…}
Graph
program
program_id
program_id
student_semester
v1
semester_id
semester_id
semester
v4
name
student_id
student_id
student
v2
v3
name
Decoder
LSTM
SELECT
name
FROM
semester
JOIN
F(v1, v3) = 1.4
F(v1, v4) = 3.3
× â1 (0.23)
× â2 (0.77)
s_j^att(v1) = 2.9 student_semester
s_j^att(v2) = 0.3 student
p_j

Figure 3: Left: DB schema and question. Middle: A graph representation of the schema. Bold nodes are tables, other nodes are columns. Dashed red (blue) edges are foreign (primary) keys edges, green edges are table-column edges. Right: Use of the schema by the decoder. For clarity, the decoder outputs tokens rather than grammar rules.

s_j^{loc} is computed for all legal schema items by multiplying the matrix $S_{\text{link}} \in \mathbb{R}^{\mathcal{V}_{\text{legal}} \times |x|}$, which contains the relevant linking scores $s_{\text{link}}(v, x_i)$, with the attention vector a_j. Thus, decoding unseen schema items is done by first attending to the question words, which are linked to the schema items.

4 Modeling Schemas with GNNs

Schema structure is informative for predicting the SQL query. Consider a table with two columns, where each is a foreign key to two other tables (`student_semester` table in Figure 3). Such a table is commonly used for describing a many-to-many relation between two other tables, which affects the output query. We now show how we represent this information in a neural parser and use it to improve predictions.

At a high-level our model has the following parts (Figure 3). (a) The schema is converted to a graph. (b) The graph is softly pruned conditioned on the input question. (c) A Graph neural network generates a representation for nodes that is aware of the global schema structure. (d) The encoder and decoder use the schema representation. We will now elaborate on each part.

Schema-to-graph To convert the schema S to a graph (Figure 3, left), we define the graph nodes as the schema items $\mathcal{V}$. We add three types of edges: for each column c_t in a table t, we add edges (c_t, t) and (t, c_t) to the edge set $\mathcal{E}_{\leftrightarrow}$ (green edges). For each foreign-primary key column pair $(c_{t_1}, c_{t_2}) \in \mathcal{F}$, we add edges (c_{t_1}, c_{t_2}) and (t_1, t_2) to the edge set $\mathcal{E}_{\rightarrow}$ and edges (c_{t_2}, c_{t_1}) and (t_2, t_1) to $\mathcal{E}_{\leftarrow}$ (dashed edges). Edge types are used by the graph neural network to capture different ways in which columns and tables relate to one another.

Question-conditioned relevance Each question refers to different parts of the schema, and thus, our representation should change conditioned on the question. For example, in Figure 3, the relation between the tables `student_semester` and `program` is irrelevant. To model that, we re-use the distribution $p_{\text{link}}(\cdot)$ from §3, and define a relevance score for a schema item v: $\rho_v = \max_i p_{\text{link}}(v \mid x_i)$ — the maximum probability of v for any word x_i. We use this score next to create a question-conditioned graph representation. Figure 3 shows relevant schema items in dark orange, and irrelevant items in light orange.

Neural graph representation To learn a node representation that considers its relevance score and the global schema structure, we use gated GNNs (Li et al., 2016). Each node v is given an initial embedding conditioned on the relevance score: $h_v^{(0)} = r_v \cdot \rho_v$. We then apply the GNN recurrence for L steps. At each step, each node re-computes its representation based on the representation of its neighbors in the previous step:

$$a_v^{(l)} = \sum_{\text{type} \in \{\rightarrow, \leftrightarrow\}} \sum_{(u,v) \in \mathcal{E}_{\text{type}}} W_{\text{type}} h_u^{l-1} + b_{\text{type}},$$

and then $h_v^{(l)}$ is computed as following, using a standard GRU (Cho et al., 2014) update:

$$h_v^{(l)} = \text{GRU}(h_v^{(l-1)}, a_v^{(l)})$$

(see Li et al. (2016) for further details).

We denote the final representation of each schema item after L steps by $\varphi_v = h_v^{(L)}$. We now show how this representation is used by the parser.

Encoder In §3, a weighted average over schema items l_i was concatenated to every word x_i. To enjoy the schema-aware representations, we compute $l_i^{\varphi} = \sum_{\tau} \sum_{v \in \mathcal{V}_{\tau}} \varphi_v p_{\text{link}}(v \mid x_i)$, which is identical to l_i, except φ_v is used instead of r_v. We concatenate l_i^{φ} to the output of the encoder h_i, so that each word is augmented with the graph structure around the schema items it is linked to.

Decoder As mentioned (§3), when a schema item v is decoded, the input in the next time step is its type $\tau(v)$. A first change is to replace $\tau(v)$ by φ_v, which has knowledge of the structure around v. A second change is a self-attention mechanism that links to the schema, which we describe next.

When scoring a schema item, its score should depend on its relation to previously decoded schema items. E.g., in Figure 3, once the table `semester` has been decoded, it is likely to be joined to a related table. We capture this intuition with a self-attention mechanism.

For each decoding step j, we denote by u_j the hidden state of the decoder, and by $\hat{J} = (i_1, \ldots, i_{|\hat{J}|})$ the list of time steps before j where a schema item has been decoded. We define the matrix $\hat{U} \in \mathbb{R}^{d \times |\hat{J}|} = [u_{i_1}, \ldots, u_{i_{|\hat{J}|}}]$, which concatenates the hidden states from all these time steps. We now compute a self-attention distribution over these time steps, and score schema items based on this distribution (Figure 3, right):

$$\begin{aligned} \hat{a}_j &= \text{softmax}(\hat{U}^T u_j) \quad \in \mathbb{R}^{|\hat{J}|}, \\ s_j^{\text{att}} &= \hat{a}_j S^{\text{att}}, \\ p_j &= \text{softmax}([s_j^{\text{glob}}; s_j^{\text{loc}} + s_j^{\text{att}}]), \end{aligned}$$

where the matrix $S^{\text{att}} \in \mathbb{R}^{|\hat{J}| \times \mathcal{V}_{\text{legal}}}$ computes a similarity between schema items that were previously decoded, and schema items that are legal according to the grammar: $S^{\text{att}}_{v_1, v_2} = F(\varphi_{v_1})^{\top} F(\varphi_{v_2})$, where $F(\cdot)$ is a feed-forward network. Thus, the score of a schema item increases, if substantial attention is placed on schema items to which it bears high similarity.

Training We maximize the log-likelihood of the gold sequence during training, and use beam-search (of size 10) at test time, similar to Krishnamurthy et al. 2017 and prior work. We run the GNN for $L = 2$ steps.

Model	Acc.	SINGLE	MULTI
SQLNET	10.9%	13.6%	3.3%
SYNTAXSQLNET	18.9%	23.1%	7.0%
NO GNN	34.9%	52.3%	14.6%
GNN	**40.7%**	52.2%	**26.8%**
- NO SELF ATTEND	38.7%	**54.5%**	20.3%
- ONLY SELF ATTEND	35.9%	47.1%	23.0%
- NO REL.	37.0%	50.4%	21.5%
GNN ORACLE REL.	54.3%	63.5%	43.7%

Table 1: Development set accuracy for all models.

5 Experiments and Results

Experimental setup We evaluate on SPIDER (Yu et al., 2018b), which contains 7,000/1,034/2,147 train/development/test examples.

We pre-process examples to remove table aliases (`AS T1/T2/...`) from the queries and use the explicit table name instead (i.e. we replace `T1.col` with `table1_name.col`), as in the majority of the cases (> 99% in SPIDER) these aliases are redundant. In addition, we add a table reference to all columns that do not have one (i.e. we replace `col` with `table_name.col`).

We use the official evaluation script from SPIDER to compute accuracy, i.e., whether the predicted query is equivalent to the gold query.

Results Our full model (GNN) obtains 39.4% accuracy on the test set, substantially higher than prior state-of-the-art (SYNTAXSQLNET), which is at 19.7%. Removing the GNN from the parser (NO GNN), which results in the parser of Krishnamurthy et al. (2017), augmented with a grammar for SQL, obtains an accuracy of 33.8%, showing the importance of encoding the schema structure.

Table 1 shows results on the development set for baselines and ablations. The first column describes accuracy on the entire dataset, and the next two columns show accuracy when partitioning examples to queries involving only one table (SINGLE) vs. more than one table (MULTI).

GNN dramatically outperforms previously published baselines SQLNET and SYNTAXSQLNET, and improves the performance of NO GNN from 34.9% to 40.7%. Importantly, using schema structure specifically improves performance on questions with multiple tables from 14.6% to 26.8%.

We ablate the major novel components of our model to assess their impact. First, we remove the self-attention component (NO SELF ATTEND). We observe that performance drops by 2 points, where SINGLE slightly improves, and MULTI drops by 6.5 points. Second, to verify that improvement is not only due to self-attention, we ablate all other uses of the GNN. Namely, We use a model identical to NO GNN, except it can access the GNN representations through the self-attention (ONLY SELF ATTEND). We observe a large drop in performance to 35.9%, showing that all components are important. Last, we ablate the relevance score by setting $\rho_v = 1$ for all schema items (NO REL.). Indeed, accuracy drops to 37.0%.

To assess the ceiling performance possible with a perfect relevance score, we run an oracle experiment, where we set $\rho_v = 1$ for all schema items that are in the gold query, and $\rho_v = 0$ for all other schema items (GNN ORACLE REL.). We see that a perfect relevance score substantially improves performance to 54.3%, indicating substantial headroom for future research.

`join` analysis For any model, we can examine the proportion of predicted queries with a `join`, where the structure of the `join` is "bad": (a) when the `join` condition clause uses the same table twice (`ON t1.column1 = t1.column2`), and (b) when the joined table are not connected through a primary-foreign key relation.

We find that NO GNN predicts such `joins` in 83.4% of the cases, while GNN does so in only 15.6% of cases. When automatically omitting from the beam candidates where condition (a) occurs, NO GNN predicts a "bad" `join` in 14.2% of the cases vs. 4.3% for GNN (total accuracy increases by 0.3% for both models). As an example, in Figure 3, s_j^{loc} scores the table `student` the highest, although it is not related to the previously decoded table `semester`. Adding the self-attention score s_j^{att} corrects this and leads to the correct `student_semester`, probably because the model learns to prefer connected tables.

6 Conclusion

We present a semantic parser that encodes the structure of the DB schema with a graph neural network, and uses this representation to make schema-aware decisions both at encoding and decoding time. We demonstrate the effectivness of this method on SPIDER, a dataset that contains complex schemas which are not seen at training time, and show substantial improvement over current state-of-the-art.

Acknowledgments

We thank Kevin Lin and Mark Neumann from Allen Institute for Artificial Intelligence for their help with the SQL grammar. This research was supported by Facebook. This work was completed in partial fulfillment for the Ph.D degree of the first author.

References

D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

J. Cheng, S. Reddy, V. Saraswat, and M. Lapata. 2017. Learning structured natural language representations for semantic parsing. In *Association for Computational Linguistics (ACL)*.

K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

N. De Cao, W. Aziz, and I. Titov. 2019. Question answering by reasoning across documents with graph convolutional networks. In *North American Association for Computational Linguistics (NAACL)*.

C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. Radev. 2018. Improving text-to-sql evaluation methodology. In *Association for Computational Linguistics (ACL)*.

M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*.

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

S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Association for Computational Linguistics (ACL)*.

J. Krishnamurthy, P. Dasigi, and M. Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. 2016. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*.

K. Lin, B. Bogin, M. Neumann, J. Berant, and M. Gardner. 2019. Grammar-based neural text-to-sql generation. *arXiv preprint arXiv:1905.13326*.

M. Rabinovich, M. Stern, and D. Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Association for Computational Linguistics (ACL)*.

D. Sorokin and I. Gurevych. 2018. Modeling semantics with gated graph neural networks for knowledge base question answering. In *International Conference on Computational Linguistics (COLING)*.

C. Xiao, M. Dymetman, and C. Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Association for Computational Linguistics (ACL)*.

P. Yin and G. Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Association for Computational Linguistics (ACL)*, pages 440–450.

T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li, and D. Radev. 2018a. SyntaxSQLNet: Syntax tree networks for complex and cross-domaintext-to-SQL task. In *Empirical Methods in Natural Language Processing (EMNLP)*.

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. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Empirical Methods in Natural Language Processing (EMNLP)*.

M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1050–1055.

L. S. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*, pages 658–666.

V. Zhong, C. Xiong, and R. Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.