Clause-Wise and Recursive Decoding for Complex and Cross-Domain Text-to-SQL Generation

Dongjun Lee
SAP Labs Korea
dongjun.lee01@sap.com

Abstract

Most deep learning approaches for text-to-SQL generation are limited to the WikiSQL dataset, which only supports very simple queries over a single table. We focus on the *Spider* dataset, a complex and cross-domain text-to-SQL task, which includes complex queries over multiple tables. In this paper, we propose a SQL clause-wise decoding neural architecture with a self-attention based database schema encoder to address the *Spider* task. Each of the clause-specific decoders consists of a set of sub-modules, which is defined by the syntax of each clause. Additionally, our model works recursively to support nested queries. When evaluated on the *Spider* dataset, our approach achieves 4.6% and 9.8% accuracy gain in the test and dev sets, respectively. In addition, we show that our model is significantly more effective at predicting complex and nested queries than previous work.

1 Introduction

Text-to-SQL generation is the task of translating a natural language question into the corresponding SQL. Recently, various deep learning approaches have been proposed based on the WikiSQL dataset (Zhong et al., 2017). However, because WikiSQL contains only very simple queries over just a single table, these approaches (Xu et al., 2017; Huang et al., 2018; Yu et al., 2018a; Dong and Lapata, 2018) cannot be applied directly to generate complex queries containing elements such as `JOIN`, `GROUP BY`, and nested queries.

To overcome this limitation, Yu et al. (2018c) introduced *Spider*, a new complex and cross-domain text-to-SQL dataset. It contains a large number of complex queries over different databases with multiple tables. It also requires a model to generalize to unseen database schema as different databases are used for training and testing. Therefore, a model should understand not only the natural language question but also the schema of the corresponding database to predict the correct SQL query.

In this paper, we propose a novel SQL-specific clause-wise decoding neural network model to address the *Spider* task. We first predict a sketch for each SQL clause (e.g., `SELECT`, `WHERE`) with text classification modules. Then, clause-specific decoders find the columns and corresponding operators based on the sketches. Our contributions are summarized as follows.

- We decompose the clause-wise SQL decoding process. We also modularize each of the clause-specific decoders into sub-modules based on the syntax of each clause. Our architecture enables the model to learn clause-dependent context and also ensures the syntactic correctness of the predicted SQL.
- Our model works recursively so that it can predict nested queries.
- We also introduce a self-attention based database schema encoder that enables our model to generalize to unseen databases.

In the experiment on the *Spider* dataset, we achieve 24.3% and 28.8% exact SQL matching accuracy on the test and dev set respectively, which outperforms the previous state-of-the-art approach (Yu et al., 2018b) by 4.6% and 9.8%. In addition, we show that our approach is significantly more effective compared to previous work at predicting not only simple SQL queries, but also complex and nested queries.

2 Related Work

Our work is related to the grammar-based constrained decoding approaches for semantic parsing (Yin and Neubig, 2017; Rabinovich et al., 2017;

Iyer et al., 2018). While their approaches are focused on general purpose code generation, we instead focus on SQL-specific grammar to address the text-to-SQL task. Our task differs from code generation in two aspects. First, it takes a database schema as an input in addition to natural language. To predict SQL correctly, a model should fully understand the relationship between the question and the schema. Second, as SQL is a non-procedural language, predictions of SQL clauses do not need to be done sequentially.

For text-to-SQL generation, several SQL-specific approaches have been proposed (Zhong et al., 2017; Xu et al., 2017; Huang et al., 2018; Yu et al., 2018a; Dong and Lapata, 2018; Yavuz et al., 2018) based on WikiSQL dataset (Zhong et al., 2017). However, all of them are limited to the specific WikiSQL SQL sketch, which only supports very simple queries. It includes only the `SELECT` and `WHERE` clauses, only a single expression in the `SELECT` clause, and works only for a single table. To predict more complex SQL queries, sequence-to-sequence (Iyer et al., 2017; Finegan-Dollak et al., 2018) and template-based (Finegan-Dollak et al., 2018; Lee et al., 2019) approaches have been proposed. However, they focused only on specific databases such as ATIS (Price, 1990) and GeoQuery (Zelle and Mooney, 1996). Because they only considered question and SQL pairs without requiring an understanding of database schema, their approaches cannot generalize to unseen databases.

SyntaxSQLNet (Yu et al., 2018b) is the first and state-of-the-art model for the *Spider* (Yu et al., 2018c), a complex and cross-domain text-to-SQL task. They proposed an SQL specific syntax tree-based decoder with SQL generation history. Our approach differs from their model in the following aspects. First, taking into account that SQL corresponds to non-procedural language, we develop a clause-specific decoder for each SQL clause, where SyntaxSQLNet predicts SQL tokens sequentially. For example, in SyntaxSQLNet, a single column prediction module works both in the `SELECT` and `WHERE` clauses, depending on the SQL decoding history. In contrast, we define and train decoding modules separately for each SQL clause to fully utilize clause-dependent context. Second, we apply sequence-to-sequence architecture to predict columns instead of using the sequence-to-set framework from SyntaxSQLNet, because correct ordering is essential for the `GROUP BY` and `ORDER BY` clauses. Finally, we introduce a self-attention mechanism (Lin et al., 2017) to efficiently encode database schema, which includes multiple tables.

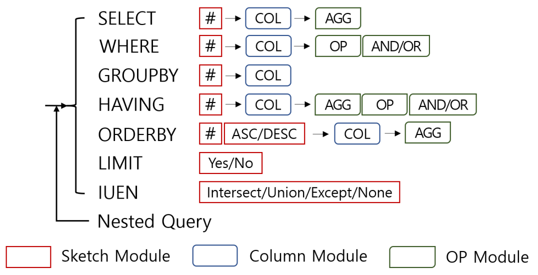

Figure 1: Clause-wise and recursive SQL generation process.

3 Methodology

We predict complex SQL clause-wisely as described in Figure 1. Each clause is predicted consecutively by at most three different types of modules (sketch, column, operator). The same architecture recursively predicts nested queries with temporal predicted SQL as an additional input.

3.1 Question and Schema Encoding

We encode a natural language question with a bi-directional LSTM. We denote $H_Q \in \mathbb{R}^{d \times |X|}$ as the question encoding, where d is the number of LSTM units and $|X|$ is the number of tokens in the question.

To encode a database schema, we consider each column in its tables as a concatenated sequence of words from the table name and column name with a separation token. (e.g., [student, `[SEP]`, first, name]). First, we apply bi-directional LSTM over this sequence for each column. Then, we apply the self-attention mechanism (Lin et al., 2017) over the LSTM outputs to form a summarized fixed-size vector for each column. For the ith column, its encoding $h_{col}^{(i)} \in \mathbb{R}^d$ is computed by a weighted sum of the LSTM output $o_{col}^{(i)} \in \mathbb{R}^{d \times |L|}$ as follows:

$$\alpha = \mathtt{softmax}(w^T \mathtt{tanh}(o_{col}^{(i)})) \quad (1)$$

$$h_{col}^{(i)} = o_{col}^{(i)} \alpha^T \quad (2)$$

where $|L|$ is the number of tokens in the column and $w \in \mathbb{R}^d$ is a trainable parameter. We denote $H_{col} = [h_{col}^{(1)}, ...h_{col}^{(|C|)}]$ as columns encoding where $|C|$ is the number of columns in the database.

3.2 Sketch Prediction

We predict the clause-wise sketch via 8 different text classification modules that include the number of SQL expressions in each clause, the presence of `LIMIT` clause, and the presence of `INTERSECT/UNION/EXCEPT` as described in Figure 1. All of them share the same model architecture but are trained separately. For the classification, we applied attention-based bi-directional LSTM following Zhou et al. (2016).

First, we compute sentence representation $r_s \in \mathbb{R}^d$ by a weighted sum of question encoding $H_Q \in \mathbb{R}^{d\times|X|}$. Then we apply the softmax classifier to choose the sketch as follows:

$$\alpha_s = \texttt{softmax}(w_s^T \texttt{tanh}(H_Q)) \quad (3)$$

$$r_s = H_Q\, \alpha_s^T \quad (4)$$

$$P_{sketch} = \texttt{softmax}(W_s r_s + b_s) \quad (5)$$

where $w_s \in \mathbb{R}^d, W_s \in \mathbb{R}^{n_s\times d}, b_s \in \mathbb{R}^{n_s}$ are trainable parameters and n_s is the number of possible sketches.

3.3 Columns and Operators Prediction

To predict columns and operators, we use the LSTM decoder with the attention mechanism (Luong et al., 2015) such that *the number of decoding steps are decided by the sketch prediction module.* We train 5 different column prediction modules separately for each SQL clause, but they share the same architecture.

In the column prediction module, the hidden state of the decoder at the t-th decoding step is computed as $d_{col}^{(t)}(\in \mathbb{R}^d) = \texttt{LSTM}(d_{col}^{(t-1)}, h_{col}^{(t-1)})$, where $h_{col}^{(t-1)} \in \mathbb{R}^d$ is an encoding of the predicted column in the previous decoding step. The context vector $r^{(t)}$ is computed by a weighted sum of question encodings $H_Q \in \mathbb{R}^{d\times|X|}$ based on attention weight as follows:

$$\alpha^{(t)} = \texttt{softmax}({d_{col}^{(t)}}^T H_Q) \quad (6)$$

$$r^{(t)} = H_Q\, {\alpha^{(t)}}^T \quad (7)$$

Then, the attentional output of the t-th decoding step $a_{col}^{(t)}$ is computed as a linear combination of $d_{col}^{(t)} \in \mathbb{R}^d$ and $r^{(t)} \in \mathbb{R}^d$ followed by `tanh` activation.

$$a_{col}^{(t)} = \texttt{tanh}(W_1 d_{col}^{(t)} + W_2 r^{(t)}) \quad (8)$$

where $W_1, W_2 \in \mathbb{R}^{d\times d}$ are trainable parameters. Finally, the probability for each column at the t-th decoding step is computed as a dot product between $a_{col}^{(t)} \in \mathbb{R}^d$ and the encoding of each column in $H_{col} \in \mathbb{R}^{d\times|C|}$ followed by softmax.

$$P_{col}^{(t)} = \texttt{softmax}({a_{col}^{(t)}}^T H_{col}) \quad (9)$$

To predict corresponding operators for each predicted column, we use a decoder of the same architecture as in the column prediction module. The only difference is that a decoder input at the t-th decoding step is an encoding of the t-th predicted column from the column prediction module.

$$d_{op}^{(t)} = \texttt{LSTM}(d_{op}^{(t-1)}, h_{col}^{(t)}) \quad (10)$$

Attentional output $a_{op}^{(t)} \in \mathbb{R}^d$ is computed identically to Eq. (8). Then, the probability for operators corresponding to the t-th predicted column is computed by the softmax classifier as follows:

$$P_{op}^{(t)} = \texttt{softmax}(W_o a_{op}^{(t)} + b_o) \quad (11)$$

where $W_o \in \mathbb{R}^{n_o\times d}$ and $b_o \in \mathbb{R}^{n_o}$ are trainable parameters and n_o is the number of possible operators.

3.4 From Clause Prediction

After the predictions of all the other clauses, we use a heuristic to generate the `FROM` clause. We first collect all the columns that appear in the predicted SQL, and then we `JOIN` tables that include these predicted columns.

3.5 Recursion for Nested Queries

To predict the presence of a sub-query, we train another module that has the same architecture as the operator prediction module. Instead of predicting corresponding operators for each column, it predicts whether each column is compared to a variable (e.g., `WHERE` age $>$ 3) or to a sub-query (e.g., `WHERE` age $>$ (`SELECT` `avg`(age) ..)). In the latter case, we add the temporal `[SUB_QUERY]` token to the corresponding location in the SQL output. Additionally, if the sketch prediction module predicts one of `INTERSECT/UNION/EXCEPT` operators, we add a `[SUB_QUERY]` token after the operator.

To predict a sub-query, our model takes the temporal generated SQL with a `[SUB_QUERY]` token as an input in addition to a natural language question with separate token `[SEP]` (e.g.,

Method	Dev Easy	Dev Medium	Dev Hard	Dev Extra Hard	Dev All	Test All
SQLNet	23.2%	8.6%	9.8%	0%	10.9%	12.4%
TypeSQL	18.8%	5.5%	4.6%	2.4%	8.0%	8.2%
SyntaxSQLNet	38.4%	15.0%	16.1%	3.5%	19.0%	19.7%
Ours	**53.2%**	**27.0%**	**20.1%**	**6.5%**	**28.8%**	**24.3%**
-rec	53.2%	27.0%	14.4%	2.9%	27.4%	-
-rec - col-att	46.4%	22.0%	12.1%	4.7%	23.4%	-
-rec -col-att -sketch	33.2%	18.6%	11.5%	4.7%	18.7%	-

Table 1: Accuracy of exact SQL matching with different hardness levels.

Method	SELECT	WHERE	GROUP BY	ORDER BY	KEYWORDS
SQLNet	46.6%	20.6%	37.6%	49.2%	62.8%
TypeSQL	43.7%	14.8%	16.9%	52.1%	67.0%
SyntaxSQLNet	55.4%	22.2%	51.4%	50.6%	73.3%
Ours	**68.7%**	**39.0%**	**63.1%**	**63.5%**	**76.5%**

Table 2: F1 scores of SQL component matching on the *dev* set.

What is ... `[SEP] SELECT ... INTERSECT [SUB_QUERY]`). This input is encoded in the same way as question encoding described in Section 3.1. Then, the rest of the SQL generation process is identical to that described in Section 3.2–3.4. After the sub-query is predicted, it replaces the `[SUB_QUERY]` token to form the final query.

4 Experiments

4.1 Experimental Setup

We evaluate our model with *Spider* (Yu et al., 2018c), a large-scale, complex and cross-domain text-to-SQL dataset. We follow the same database split as Yu et al. (2018c), which ensures that any database schema that appears in the training set does not appear in the dev or test set. Through this split, we examine how well our model can be generalized to unseen databases. Because the test set is not opened to the public, we use the *dev* set for the ablation analysis. For the evaluation metrics, we use 1) accuracy of exact SQL matching and 2) F1 score of SQL component matching, proposed by (Yu et al., 2018c). We also follow their query hardness criteria to understand the model performance on different levels of queries. Our model and all the baseline models are trained based on only the *Spider* dataset without data augmentation.

4.2 Model Configuration

We use the same hyperparameters for every module. For the word embedding, we apply deep contextualized word representations (ELMO) from Peters et al. (2018) and allow them to be fine-tuned during the training. For the question and column encoders, we use a 1-layer 512-unit bi-directional LSTM. For the decoders in the columns and operators prediction modules, we use a 1-layer 1024-unit uni-directional LSTM. For the training, we use Adam optimizer (Kingma and Ba, 2014) with a learning rate of 1e-4 and use early stopping with 50 epochs. Additionally, we use dropout (Hinton et al., 2012) with a rate of 0.2 for the regularization.

4.3 Result and Analysis

Table 1 shows the exact SQL matching accuracy of our model and previous models. We achieve 24.3% and 28.8% on the test and dev sets respectively, which outperforms the previous best model SyntaxSQLNet (Yu et al., 2018b) by 4.6% and 9.8%. Moreover, our model outperforms previous models on all different query hardness levels.

To examine how each technique contributes to the performance, we conduct an ablation analysis of three aspects: 1) without recursion, 2) without self-attention for database schema encoding, and 3) without sketch prediction modules that decide the number of decoding steps. Without recursive sub-query generation, the accuracy drops by 5.7% and 3.6% for hard and extra hard queries, respectively. This result shows that the recursion we use enables the model to predict nested queries. When using the final LSTM hidden state as in Yu et al. (2018b) instead of using self-attention for schema encoding, the accuracy drops by 4.0% on all queries. Finally, when using only an encoder-decoder architecture without sketch generation for columns prediction, the accuracy drops by 4.7%.

For the component matching result for each SQL clause, our model outperforms previous approaches for all of the SQL components by a significant margin, as shown in Table 2. Examples of predicted SQL from different models are shown in Appendix A.

5 Conclusion

In this paper, we propose a recursive and SQL clause-wise decoding neural architecture to address the complex and cross-domain text-to-SQL task. We evaluate our model with the *Spider* dataset, and the experimental result shows that our model significantly outperforms previous work for generating not only simple queries, but also complex and nested queries.

Acknowledgments

We thank Yongsik Lee, Jaesik Yoon, and Donghun Lee (SAP) for their reviews and support. We also thank professor Sungroh Yoon, Jongyun Song, and Taeuk Kim (Seoul National University) for their insightful feedback and three anonymous reviewers for their helpful comments.

References

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

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 351–360.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580.*

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, volume 2, pages 732–738.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760.*

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Dongjun Lee, Jaesik Yoon, Jongyun Song, Sanggil Lee, and Sungroh Yoon. 2019. One-shot learning for text-to-sql generation. *arXiv preprint arXiv:1905.11499.*

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130.*

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 2227–2237.

Patti J Price. 1990. Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990.*

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436.*

Semih Yavuz, Izzeddin Gur, Yu Su, and Xifeng Yan. 2018. What it takes to achieve 100% condition accuracy on wikisql. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1702–1711.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers), pages 440–450.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, volume 2, pages 588–594.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018b. Syntaxsqlnet: Syntax tree networks for complex and cross-domain text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018c. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

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

Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 207–212.

A Sample SQL Predictions

In Table 3, we show some examples of predicted SQL queries from different models. We compare the result of our model with two of previous state-of-the-art models: SyntaxSQLNet (Yu et al., 2018b) and the modified version of SQLNet (Xu et al., 2017) by Yu et al. (2018c) to support complex SQL queries.

hardness	type	description
easy	NL	What are the names of all the countries that became independent after 1950?
	Truth	SELECT Name FROM country WHERE IndepYear > 1950
	Ours	SELECT Name FROM country WHERE IndepYear > "[VAR]"
	Syntax	SELECT Name FROM country WHERE GovernmentForm = "[VAR]"
	SQLNet	SELECT T1.Name FROM city as T1 JOIN country as T2 WHERE T2.Population > "[VAR]"
medium	NL	Which city and country is the Alton airport at?
	Truth	SELECT City, Country FROM airports WHERE AirportName = "Alton"
	Ours	SELECT City, Country FROM airports WHERE AirportName = "[VAR]"
	Syntax	SELECT Country, City FROM airports WHERE Country = "[VAR]"
	SQLNet	SELECT T1.City, T2.DestAirport FROM airports as T1 JOIN flights as T2
medium	NL	List the names of poker players ordered by the final tables made in ascending order.
	Truth	SELECT T1.Name FROM people as T1 JOIN poker_player as T2 ORDER BY T2.Final_Table_Made
	Ours	SELECT T1.Name FROM people as T1 JOIN poker_player as T2 ORDER BY T2.Final_Table_Made ASC
	Syntax	SELECT T2.Name FROM poker_player as T1 JOIN people as T2 ORDER BY T1.Earnings ASC
	SQLNet	SELECT Name FROM people ORDER BY Birth_Date ASC
medium	NL	How much does the most recent treatment cost?
	Truth	SELECT cost_of_treatment FROM Treatments ORDER BY date_of_treatment DESC LIMIT 1
	Ours	SELECT cost_of_treatment FROM Treatments ORDER BY cost_of_treatment DESC LIMIT "[VAR]"
	Syntax	SELECT cost_of_treatment FROM Treatments ORDER BY cost_of_treatment ASC LIMIT "[VAR]"
	SQLNet	SELECT T1.charge_amount FROM Charges as T1 JOIN Dogs as T2 ORDER BY date_adopted DESC LIMIT "[VAR]"
hard	NL	List the names of teachers who have not been arranged to teach courses.
	Truth	SELECT Name FROM teacher WHERE Teacher_id NOT IN (SELECT Teacher_id FROM course_arrange)
	Ours	SELECT Name FROM teacher WHERE Teacher_id NOT IN (SELECT Teacher_id FROM course_arrange)
	Syntax	SELECT Name FROM teacher
	SQLNet	SELECT Name FROM teacher
hard	NL	Which cities do more than one employee under age 30 come from?
	Truth	SELECT city FROM employee WHERE age < 30 GROUP BY city HAVING count(*) > 1
	Ours	SELECT city FROM employee WHERE age < "[VAR]" GROUP BY city HAVING count(*) > "[VAR]"
	Syntax	SELECT city FROM employee WHERE age > "[VAR]"
	SQLNet	SELECT T1.city FROM employee as T1 JOIN hiring as T2 JOIN shop as T3 WHERE T3.District > "[VAR]" GROUP BY T1.city HAVING count(*) > "[VAR]"
hard	NL	What is the document id with least number of paragraphs?
	Truth	SELECT document_id FROM Paragraphs GROUP BY document_id ORDER BY count(*) LIMIT 1
	Ours	SELECT document_id FROM Documents GROUP BY document_id ORDER BY count(*) ASC LIMIT "[VAR]"
	Syntax	SELECT document_id FROM Documents GROUP BY document_id ORDER BY count(*) ASC LIMIT "[VAR]" HAVING count(*) >= "[VAR]"
	SQLNet	SELECT template_id FROM Templates GROUP BY template_id HAVING sum(*) NOT "[VAR]" ORDER BY count(*) ASC LIMIT "[VAR]"
extra	NL	How many dogs have not gone through any treatment?
	Truth	SELECT count(*) FROM Dogs WHERE dog_id NOT IN (SELECT dog_id FROM Treatments)
	Ours	SELECT count(*) FROM Dogs WHERE dog_id NOT IN (SELECT dog_id FROM Treatments)
	Syntax	SELECT count(*) FROM Charges WHERE charge_id NOT IN (SELECT charge_id FROM Charges)
	SQLNet	SELECT count(*) FROM Dogs WHERE dog_id IN "[VAR]"
extra	NL	What is the name of the high schooler who has the greatest number of friends?
	Truth	SELECT T2.name FROM Friend as T1 JOIN Highschooler as T2 GROUP BY T1.student_id ORDER BY count(*) DESC LIMIT 1
	Ours	SELECT T1.name FROM Highschooler as T1 JOIN Friend as T2 GROUP BY T2.student_id ORDER BY count(*) DESC LIMIT 1
	Syntax	SELECT name FROM Highschooler ORDER BY grade DESC LIMIT 1
	SQLNet	SELECT T1.name FROM Friend as T1 JOIN Friend as T2 GROUP BY T2.student_id ORDER BY * DESC LIMIT 1

Table 3: Sample SQL predictions by our model and previous state-of-the-art models on the dev split. `NL` denotes the natural language question and `Truth` denotes the corresponding ground truth SQL query. `Ours`, `Syntax`, and `SQLNet` denotes the SQL predictions from our model, SyntaxSQLNet (Yu et al., 2018b), and modified SQLNet (Xu et al., 2017) by Yu et al. (2018c), respectively.