Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions

Rui Zhang[†] **Tao Yu**[†] **He Yang Er**[†]
Sungrok Shim[†] **Eric Xue**[†] **Xi Victoria Lin**[¶] **Tianze Shi**[§]
Caiming Xiong[¶] **Richard Socher**[¶] **Dragomir Radev**[†]
[†] Yale University [¶] Salesforce Research [§] Cornell University
{r.zhang, tao.yu, dragomir.radev}@yale.edu
{xilin, cxiong, rsocher}@salesforce.com

Abstract

We focus on the cross-domain context-dependent text-to-SQL generation task. Based on the observation that adjacent natural language questions are often linguistically dependent and their corresponding SQL queries tend to overlap, we utilize the interaction history by editing the previous predicted query to improve the generation quality. Our editing mechanism views SQL as sequences and reuses generation results at the token level in a simple manner. It is flexible to change individual tokens and robust to error propagation. Furthermore, to deal with complex table structures in different domains, we employ an utterance-table encoder and a table-aware decoder to incorporate the context of the user utterance and the table schema. We evaluate our approach on the SParC dataset and demonstrate the benefit of editing compared with the state-of-the-art baselines which generate SQL from scratch. Our code is available at https://github.com/ryanzhumich/sparc_atis_pytorch.

1 Introduction

Generating SQL queries from user utterances is an important task to help end users acquire information from databases. In a real-world application, users often access information in a multi-turn interaction with the system by asking a sequence of related questions. As the interaction proceeds, the user often makes reference to the relevant mentions in the history or omits previously conveyed information assuming it is known to the system.

Therefore, in the context-dependent scenario, the contextual history is crucial to understand the follow-up questions from users, and the system often needs to reproduce partial sequences generated in previous turns. Recently, Suhr et al. (2018) proposes a context-dependent text-to-SQL model including an interaction-level encoder and an attention mechanism over previous utterances. To reuse what has been generated, they propose to copy complete segments from the previous query. While their model is successful to reason about explicit and implicit references, it does not need to explicitly address different database schemas because the ATIS contains only the flight-booking domain. Furthermore, the model is confined to copy whole segments which are extracted by a rule-based procedure, limiting its capacity to utilize the previous query when only one or a few tokens are changed in the segment.

To exploit the correlation between sequentially generated queries and generalize the system to different domains, in this paper, we study an editing-based approach for cross-domain context-dependent text-to-SQL generation task. We propose query generation by editing the query in the previous turn. To this end, we first encode the previous query as a sequence of tokens, and the decoder computes a switch to change it at the token level. This sequence editing mechanism models token-level changes and is thus robust to error propagation. Furthermore, to capture the user utterance and the complex database schemas in different domains, we use an utterance-table encoder based on BERT to jointly encode the user utterance and column headers with co-attention, and adopt a table-aware decoder to perform SQL generation with attentions over both the user utterance and column headers.

We evaluate our model on SParC (Yu et al., 2019b), a new large-scale dataset for cross-domain semantic parsing in context consisting of coherent question sequences annotated with SQL queries over 200 databases in 138 domains. Experiment results show that by generating from the previous query, our model delivers an improvement of 7% question match accuracy and 11% interaction

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 5338–5349,
Hong Kong, China, November 3–7, 2019.

	Context	Cross-Domain	Interaction (train / dev / test)	Question	Turn	Database	Table	Q. Length	Q. Vocab
Spider	✗	✓	11,840 (8,659 / 1,034 / 2,147)	11,840	1.0	200	1020	13.4	4,818
ATIS	✓	✗	1,658 (1,148 / 380 / 130)	11,653	7.0	1	27	10.2	1,582
SParC	✓	✓	4,298 (3,034 / 422 / 842)	12,726	3.0	200	1020	8.1	3,794

Table 1: Dataset Statistics.

	WHERE	AGG	GROUP	ORDER	HAVING	SET	JOIN	Nested
Spider	55.2	51.7	24.0	21.5	6.7	5.8	42.9	15.7
ATIS	100	16.6	0.3	0	0	0	96.6	96.6
SParC	42.8	39.8	20.1	17.0	4.7	3.5	35.5	5.7

Table 2: % of SQL queries that contain a particular SQL component.

Database: student dormitory containing 5 tables.

Goal: Find the first and last names of the students who are living in the dorms that have a TV Lounge as an amenity.

Q1: How many dorms have a TV Lounge?

S1:
```
SELECT COUNT(*) FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2.dormid JOIN dorm_amenity AS T3
ON T2.amenid = T3.amenid WHERE T3.amenity_name = 'TV Lounge'
```

Q2: What is the total capacity of these dorms?

S2:
```
SELECT SUM(T1.student_capacity) FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2.dormid JOIN
dorm_amenity AS T3 ON T2.amenid = T3.amenid WHERE T3.amenity_name = 'TV Lounge'
```

Q3: How many students are living there?

S3:
```
SELECT COUNT(*) FROM student AS T1 JOIN lives_in AS T2 ON T1.stuid = T2.stuid WHERE T2.dormid
IN (SELECT T3.dormid FROM has_amenity AS T3 JOIN dorm_amenity AS T4 ON T3.amenid = T4.amenid
WHERE T4.amenity_name = 'TV Lounge')
```

Q4: Please show their first and last names.

S4:
```
SELECT T1.fname, T1.lname FROM student AS T1 JOIN lives_in AS T2 ON T1.stuid = T2.stuid WHERE T2.dormid
IN (SELECT T3.dormid FROM has_amenity AS T3 JOIN dorm_amenity AS T4 ON T3.amenid = T4.amenid
WHERE T4.amenity_name = 'TV Lounge')
```

Table 3: SParC example.

match accuracy over the previous state-of-the-art. Further analysis shows that our editing approach is more robust to error propagation than copying segments, and the improvement becomes more significant if the basic text-to-SQL generation accuracy (without editing) improves.

2 Cross-Domain Context-Depencent Semantic Parsing

2.1 Datasets

We use SParC [1] (Yu et al., 2019b), a large-scale cross-domain context-dependent semantic parsing dataset with SQL labels, as our main evaluation benchmark. A SParC example is shown in Table 3. We also report performance on ATIS (Hemphill et al., 1990; Dahl et al., 1994a) for direct comparison to Suhr et al. (2018). In addition, we evaluate the cross-domain context-independent text-to-SQL ability of our model on Spider[2] (Yu et al., 2018c), which SParC is built on.

We summarize and compare the data statistics in Table 1 and Table 2. While the ATIS dataset has been extensively studied, it is limited to a particular domain. By contrast, SParC is both context-dependent and cross-domain. Each interaction in SParC is constructed using a question in Spider as the interaction goal, where the annotator asks inter-related questions to obtain information that completes the goal. SParC contains interactions over 200 databases and it follows the same database split as Spider where each database appears only in one of train, dev and test sets. In summary, SParC introduces new challenges to context-dependent text-to-SQL because it (1) contains more complex context dependencies, (2) has greater semantic coverage, and (3) adopts a cross-domain task setting.

2.2 Task Formulation

Let X denote a natural language utterance and Y denote the corresponding SQL query. Context-

[1] https://yale-lily.github.io/sparc

[2] https://yale-lily.github.io/spider

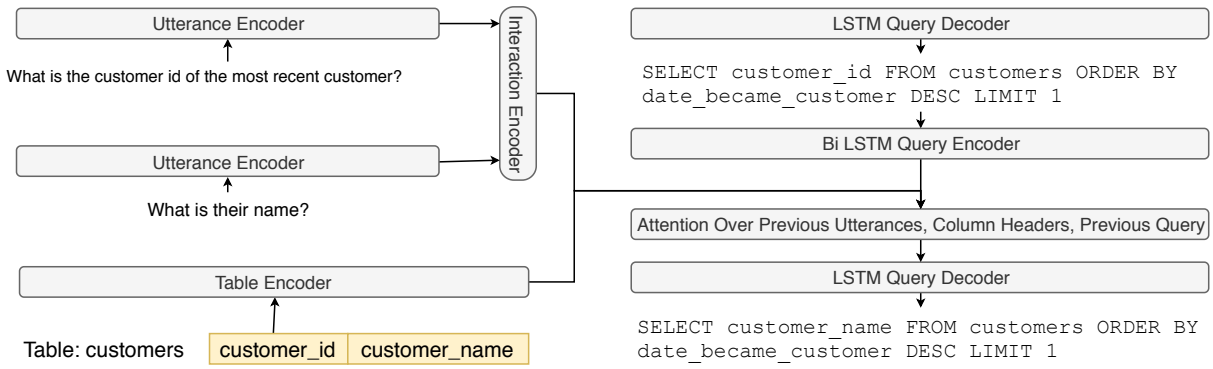


Figure 1: Model architecture of editing the previous query with attentions to the user utterances, the table schema, and the previously generated query.

independent semantic parsing considers individual (X, Y) pairs and maps X to Y. In context-dependent semantic parsing, we consider an interaction I consisting of n utterance-query pairs in a sequence:

$$I = [(X_i, Y_i)]_{i=1}^{n}$$

At each turn t, the goal is to generate Y_t given the current utterance X_t and the interaction history

$$[(X_1, Y_1), (X_2, Y_2), \ldots, (X_{t-1}, Y_{t-1})]$$

Furthermore, in the cross-domain setting, each interaction is grounded to a different database. Therefore, the model is also given the schema of the current database as an input. We consider relational databases with multiple tables, and each table contains multiple column headers:

$$T = [c_1, c_2, \ldots, c_l, \ldots, c_m]$$

where m is the number of column headers, and each c_l consists of multiple words including its table name and column name (§ 3.1).

3 Methodology

We employ an encoder-decoder architecture with attention mechanisms (Sutskever et al., 2014; Luong et al., 2015) as illustrated in Figure 1. The framework consists of (1) an utterance-table encoder to explicitly encode the user utterance and table schema at each turn, (2) A turn attention incorporating the recent history for decoding, (3) a table-aware decoder taking into account the context of the utterance, the table schema, and the previously generated query to make editing decisions.

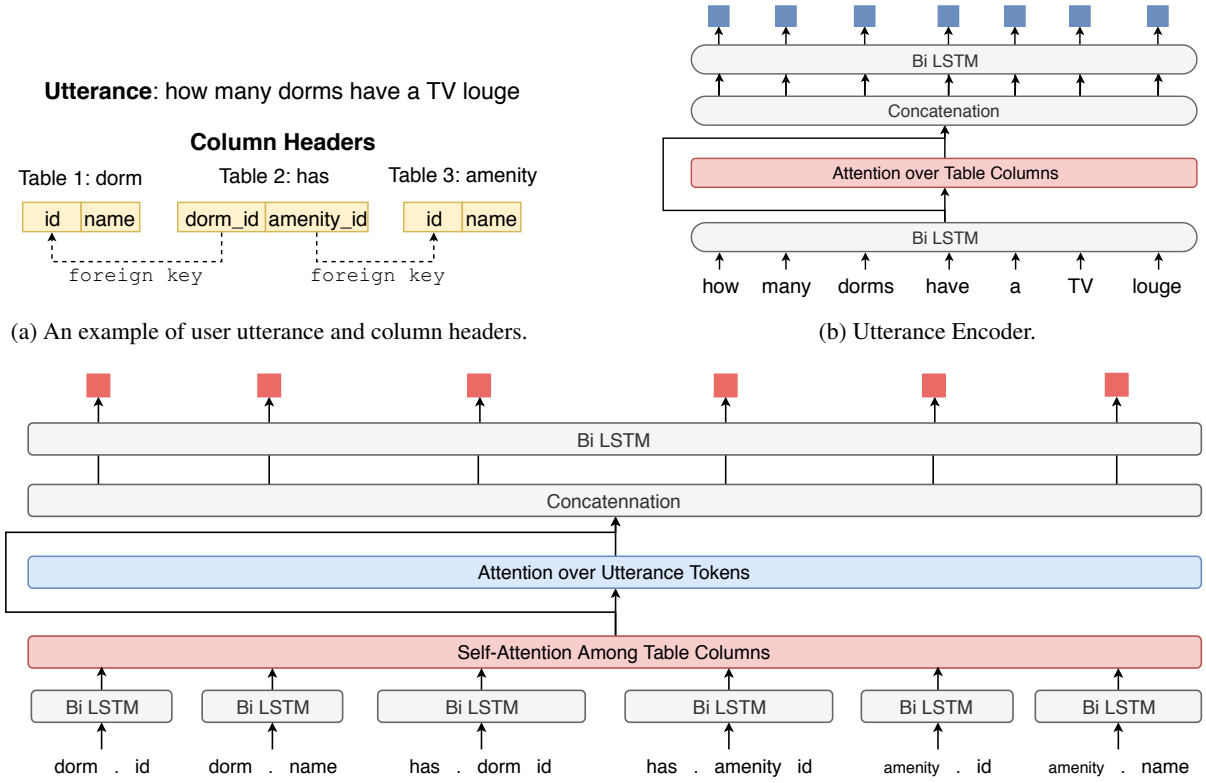
3.1 Utterance-Table Encoder

An effective encoder captures the meaning of user utterances, the structure of table schema, and the relationship between the two. To this end, we build an utterance-table encoder with co-attention between the two as illustrated in Figure 2.

Figure 2b shows the utterance encoder. For the user utterance at each turn, we first use a bi-LSTM to encode utterance tokens. The bi-LSTM hidden state is fed into a dot-product attention layer (Luong et al., 2015) over the column header embeddings. For each utterance token embedding, we get an attention weighted average of the column header embeddings to obtain the most relevant columns (Dong and Lapata, 2018). We then concatenate the bi-LSTM hidden state and the column attention vector, and use a second layer bi-LSTM to generate the utterance token embedding $\mathbf{h}^E$.

Figure 2c shows the table encoder. For each column header, we concatenate its table name and its column name separated by a special dot token (i.e., `table name . column name`). Each column header is processed by a bi-LSTM layer. To better capture the internal structure of the table schemas (e.g., foreign key), we then employ a self-attention (Vaswani et al., 2017) among all column headers. We then use an attention layer to capture the relationship between the utterance and the table schema. We concatenate the self-attention vector and the utterance attention vector, and use a second layer bi-LSTM to generate the column header embedding $\mathbf{h}^C$.

Note that the two embeddings depend on each other due to the co-attention, and thus the column header representation changes across different utterances in a single interaction.



(a) An example of user utterance and column headers.

(b) Utterance Encoder.

(c) Table Encoder.

Figure 2: Utterance-Table Encoder for the example in (a).

Utterance-Table BERT Embedding. We consider two options as the input to the first layer bi-LSTM. The first choice is the pretrained word embedding. Second, we also consider the contextualized word embedding based on BERT (Devlin et al., 2019). To be specific, we follow Hwang et al. (2019) to concatenate the user utterance and all the column headers in a single sequence separated by the `[SEP]` token:

$$\texttt{[CLS]}, X_i, \texttt{[SEP]}, c_1, \texttt{[SEP]}, \ldots, c_m, \texttt{[SEP]}$$

This sequence is fed into the pretrained BERT model whose hidden states at the last layer is used as the input embedding.

3.2 Interaction Encoder with Turn Attention

To capture the information across different utterances, we use an interaction-level encoder (Suhr et al., 2018) on top of the utterance-level encoder. At each turn, we use the hidden state at the last time step from the utterance-level encoder as the utterance encoding. This is the input to a unidirectional LSTM interaction encoder:

$$\begin{aligned} \mathbf{h}_i^U &= \mathbf{h}_{i,|X_i|}^E \\ \mathbf{h}_{i+1}^I &= \text{LSTM}^I(\mathbf{h}_i^U, \mathbf{h}_i^I) \end{aligned}$$

The hidden state of this interaction encoder $\mathbf{h}^I$ encodes the history as the interaction proceeds.

Turn Attention When issuing the current utterance, the user may omit or explicitly refer to the previously mentioned information. To this end, we adopt the turn attention mechanism to capture correlation between the current utterance and the utterance(s) at specific turn(s). At the current turn t, we compute the turn attention by the dot-product attention between the current utterance and previous utterances in the history, and then add the weighted average of previous utterance embeddings to the current utterance embedding:

$$\begin{aligned} s_i &= \mathbf{h}_t^U \mathbf{W}_{\text{turn-att}} \mathbf{h}_i^U \\ \alpha^{\text{turn}} &= \text{softmax}(s) \\ \mathbf{c}_t^{\text{turn}} &= \mathbf{h}_t^U + \sum_{i=1}^{t-1} \alpha_i^{\text{turn}} \times \mathbf{h}_i^U \end{aligned} \tag{1}$$

The $\mathbf{c}_t^{\text{turn}}$ summarizes the context information and the current user query and will be used as the initial decoder state as described in the following.

3.3 Table-aware Decoder

We use an LSTM decoder with attention to generate SQL queries by incorporating the interaction

history, the current user utterance, and the table schema.

Denote the decoding step as k, we provide the decoder input as a concatenation of the embedding of SQL query token $\mathbf{q}_k$ and a context vector $\mathbf{c}_k$:

$$\mathbf{h}^D_{k+1} = \text{LSTM}^D([\mathbf{q}_k; \mathbf{c}_k], \mathbf{h}^D_k)$$

where $\mathbf{h}^D$ is the hidden state of the decoder LSTM^D, and the hidden state $\mathbf{h}^D_0$ is initialized by $\mathbf{c}^{\text{turn}}_t$. When the query token is a SQL keyword, $\mathbf{q}_k$ is a learned embedding; when it is a column header, we use the column header embedding given by the table-utterance encoder as $\mathbf{q}_k$. The context vector $\mathbf{c}_k$ is described below.

Context Vector with the Table and User Utterance. The context vector consists of attentions to both the table and the user utterance. First, at each step k, the decoder computes the attention between the decoder hidden state and the column header embedding.

$$\begin{aligned} s_l &= \mathbf{h}^D_k \mathbf{W}_{\text{column-att}} \mathbf{h}^C_l \\ \alpha^{\text{column}} &= \text{softmax}(s) \\ \mathbf{c}^{\text{column}}_k &= \sum_l \alpha^{\text{column}}_l \times \mathbf{h}^C_l \end{aligned} \tag{2}$$

where l is the index of column headers and $\mathbf{h}^C_l$ is its embedding. Second, it also computes the attention between the decoder hidden state and the utterance token embeddings:

$$\begin{aligned} s_{i,j} &= \mathbf{h}^D_k \mathbf{W}_{\text{utterance-att}} \mathbf{h}^E_{i,j} \\ \alpha^{\text{utterance}} &= \text{softmax}(s) \\ \mathbf{c}^{\text{token}}_k &= \sum_{i,j} \alpha^{\text{utterance}}_{i,j} \times \mathbf{h}^E_{i,j} \end{aligned} \tag{3}$$

where i is the turn index, j is the token index, and $\mathbf{h}^E_{i,j}$ is the token embedding for the j-th token of i-th utterance. The context vector $\mathbf{c}_k$ is a concatenation of the two:

$$\mathbf{c}_k = [\mathbf{c}^{\text{column}}_k; \mathbf{c}^{\text{token}}_k]$$

Output Distribution. In the output layer, our decoder chooses to generate a SQL keyword (e.g., `SELECT`, `WHERE`, `GROUP BY`, `ORDER BY`) or a column header. This is critical for the cross-domain setting where the table schema changes across different examples. To achieve this, we use separate layers to score SQL keywords and column headers, and finally use the softmax operation

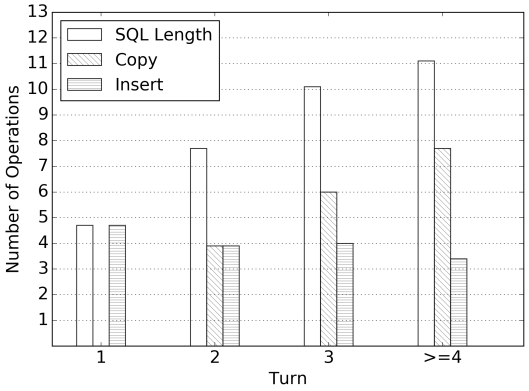


Figure 3: Number of operations at different turns.

to generate the output probability distribution:

$$\begin{aligned} \mathbf{o}_k &= \tanh([\mathbf{h}^D_k; \mathbf{c}_k]\mathbf{W}_o) \\ \mathbf{m}^{\text{SQL}} &= \mathbf{o}_k \mathbf{W}_{\text{SQL}} + \mathbf{b}_{\text{SQL}} \\ \mathbf{m}^{\text{column}} &= \mathbf{o}_k \mathbf{W}_{\text{column}} \mathbf{h}^C \\ P(y_k) &= \text{softmax}([\mathbf{m}^{\text{SQL}}; \mathbf{m}^{\text{column}}]) \end{aligned} \tag{4}$$

3.4 Query Editing Mechanism

In an interaction with the system, the user often asks a sequence of closely related questions to complete the final query goal. Therefore, the query generated for the current turn often overlaps significantly with the previous ones.

To empirically verify the usefulness of leveraging the previous query, we consider the process of generating the current query by applying copy and insert operations to the previous query[3]. Figure 3 shows the SQL query length and the number of copy and insert operations at different turns. As the interaction proceeds, the user question becomes more complicated as it requires longer SQL query to answer. However, more query tokens overlap with the previous query, and thus the number of new tokens remains small at the third turn and beyond.

Based on this observation, we extend our tableware decoder with a query editing mechanism. We first encode the previous query using another bi-LSTM, and its hidden states are the query token embeddings $\mathbf{h}^Q_{i,j'}$ (i.e., the j'-th token of the i-th query). We then extend the context vector with the attention to the previous query:

$$\mathbf{c}_k = [\mathbf{c}^{\text{column}}_k; \mathbf{c}^{\text{token}}_k; \mathbf{c}^{\text{query}}_k]$$

where $\mathbf{c}^{\text{query}}_k$ is produced by an attention to query tokens $\mathbf{h}^Q_{i,j'}$ in the same form as Equation 3.

[3]We use a diffing algorithm from https://github.com/paulgb/simplediff

At each decoding step, we predict a switch p_{copy} to decide if we need copy from the previous query or insert a new token.

$$\begin{aligned} p_{\text{copy}} &= \sigma(\mathbf{c}_k \mathbf{W}_{\text{copy}} + \mathbf{b}_{\text{copy}}) \\ p_{\text{insert}} &= 1 - p_{\text{copy}} \end{aligned} \tag{5}$$

Then, we use a separate layer to score the query tokens at turn $t-1$, and the output distribution is modified as the following to take into account the editing probability:

$$\begin{aligned} P_{\text{prev\_SQL}} &= \text{softmax}(\mathbf{o}_k \mathbf{W}_{\text{prev\_SQL}} \mathbf{h}_{t-1}^{Q}) \\ \mathbf{m}^{\text{SQL}} &= \mathbf{o}_k \mathbf{W}_{\text{SQL}} + \mathbf{b}_{\text{SQL}} \\ \mathbf{m}^{\text{column}} &= \mathbf{o}_k \mathbf{W}_{\text{column}} \mathbf{h}^{C} \\ P_{\text{SQL} \bigcup \text{column}} &= \text{softmax}([\mathbf{m}^{\text{SQL}}; \mathbf{m}^{\text{column}}]) \\ P(y_k) &= p_{\text{copy}} \cdot P_{\text{prev\_SQL}}(y_k \in \text{prev\_SQL}) \\ &+ p_{\text{insert}} \cdot P_{\text{SQL} \bigcup \text{column}}(y_k \in \text{SQL} \bigcup \text{column}) \end{aligned} \tag{6}$$

While the copy mechanism has been introduced by Gu et al. (2016) and See et al. (2017), they focus on summarization or response generation applications by copying from the source sentences. By contrast, our focus is on editing the previously generated query while incorporating the context of user utterances and table schemas.

4 Related Work

Semantic parsing is the task of mapping natural language sentences into formal representations. It has been studied for decades including using linguistically-motivated compositional representations, such as logical forms (Zelle and Mooney, 1996; Clarke et al., 2010) and lambda calculus (Zettlemoyer and Collins, 2005; Artzi and Zettlemoyer, 2011), and using executable programs, such as SQL queries (Miller et al., 1996; Zhong et al., 2017) and other general-purpose programming languages (Yin and Neubig, 2017; Iyer et al., 2018). Most of the early studies worked on a few domains and small datasets such as GeoQuery (Zelle and Mooney, 1996) and Overnight (Wang et al., 2015).

Recently, large and cross-domain text-to-SQL datasets such as WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018c) have received an increasing amount of attention as many data-driven neural approaches achieve promising results (Dong and Lapata, 2016; Su and Yan, 2017; Iyer et al., 2017; Xu et al., 2017; Finegan-Dollak et al., 2018; Yu et al., 2018a; Huang et al., 2018; Dong and Lapata, 2018; Sun et al., 2018; Gur et al., 2018; Guo et al., 2018; Yavuz et al., 2018; Shi et al., 2018). Most of them still focus on *context-independent* semantic parsing by converting single-turn questions into executable queries.

Relatively less effort has been devoted to *context-dependent* semantic parsing on datasets including ATIS (Hemphill et al., 1990; Dahl et al., 1994b), SpaceBook (Vlachos and Clark, 2014), SCONE (Long et al., 2016; Guu et al., 2017; Fried et al., 2018; Suhr and Artzi, 2018; Huang et al., 2019), SequentialQA (Iyyer et al., 2017), SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a). On ATIS, Miller et al. (1996) maps utterances to semantic frames which are then mapped to SQL queries; Zettlemoyer and Collins (2009) starts with context-independent Combinatory Categorial Grammar (CCG) parsing and then resolves references to generate lambda-calculus logical forms for sequences of sentences. The most relevant to our work is Suhr et al. (2018), who generate ATIS SQL queries from interactions by incorporating history with an interaction-level encoder and copying segments of previously generated queries. Furthermore, SCONE contains three domains using stack- or list-like elements and most queries include a single binary predicate. SequentialQA is created by decomposing some complicated questions in WikiTableQuestions (Pasupat and Liang, 2015). Since both SCONE and SequentialQA are annotated with only denotations but not query labels, they don't include many questions with rich semantic and contextual types. For example, SequentialQA (Iyyer et al., 2017) requires that the answer to follow-up questions must be a subset of previous answers, and most of the questions can be answered by simple SQL queries with `SELECT` and `WHERE` clauses.

Concurrent with our work, Yu et al. (2019a) introduced CoSQL, a large-scale cross-domain conversational text-to-SQL corpus collected under the Wizard-of-Oz setting. Each dialogue in CoSQL simulates a DB querying scenario with a crowd worker as a user and a college computer science student who is familiar with SQL as an expert. Question-SQL pairs in CoSQL reflect greater diversity in user backgrounds compared to other corpora and involve frequent changes in user intent between pairs or ambiguous questions that require user clarification. These features pose new chal-

lenges for text-to-SQL systems.

Our work is also related to recently proposed approaches to code generation by editing (Hayati et al., 2018; Yin et al., 2019; Hashimoto et al., 2018). While they follow the framework of generating code by editing the relevant examples retrieved from training data, we focus on a context-dependent setting where we generate queries from the previous query predicted by the system itself.

5 Experimental Results

5.1 Metrics

On both Spider and SParC, we use the exact set match accuracy between the gold and the predicted queries [4]. To avoid ordering issues, instead of using simple string matching, Yu et al. (2018c) decompose predicted queries into different SQL clauses such as `SELECT`, `WHERE`, `GROUP BY`, and `ORDER BY` and compute scores for each clause using set matching separately. On SparC, we report two metrics: question match accuracy which is the score average over all questions and interaction match accuracy which is average over all interactions.

5.2 Baselines

SParC. We compare with the two baseline models released by Yu et al. (2019b).

(1) Context-dependent Seq2Seq (CD-Seq2Seq): This model is adapted from Suhr et al. (2018). The original model was developed for ATIS and does not take the database schema as input hence cannot generalize well across domains. Yu et al. (2019b) adapt it to perform context-dependent SQL generation in multiple domains by adding a bi-LSTM database schema encoder which takes bag-of-words representations of column headers as input. They also modify the decoder to select between a SQL keyword or a column header.

(2) SyntaxSQL-con: This is adapted from the original context-agnostic SyntaxSQLNet (Yu et al., 2018b) by using bi-LSTMs to encode the interaction history including the utterance and the associated SQL query response. It also employs a column attention mechanism to compute representations of the previous question and SQL query.

Spider. We compare with the results as reported in Yu et al. (2018b). Furthermore, we also include recent results from Lee (2019) who propose to use recursive decoding procedure, Bogin et al. (2019) introducing graph neural networks for encoding schemas, and Guo et al. (2019) who achieve state-of-the-art performance by using an intermediate representation to bridge natural language questions and SQL queries.

	Dev Set	Test Set
SQLNet (Xu et al., 2017)	10.9	12.4
SyntaxSQLNet (Yu et al., 2018b)	18.9	19.7
+data augmentation (Yu et al., 2018b)	24.8	27.2
Lee (2019)	28.5	24.3
GNN (Bogin et al., 2019)	40.7	39.4
IRNet (Guo et al., 2019)	53.2	46.7
IRNet (BERT) (Guo et al., 2019)	61.9	54.7
Ours	36.4	32.9
+ utterance-table BERT Embedding	57.6	53.4

Table 4: Spider results on dev set and test set.

5.3 Implementation Details

Our model is implemented in PyTorch (Paszke et al., 2017). We use pretrained 300-dimensional GloVe (Pennington et al., 2014) word embedding. All LSTM layers have 300 hidden size, and we use 1 layer for encoder LSTMs, and 2 layers for decoder LSTMs. We use the ADAM optimizer (Kingma and Ba, 2015) to minimize the token-level cross-entropy loss with a batch size of 16. Model parameters are randomly initialized from a uniform distribution $U[-0.1, 0.1]$. The main model has an initial learning rate of 0.001 and it will be multiplied by 0.8 if the validation loss increases compared with the previous epoch. When using BERT instead of GloVe, we use the pretrained small uncased BERT model with 768 hidden size[5], and we fine tune it with a separate constant learning rate of 0.00001. The training typically converges in 10 epochs.

5.4 Overall Results

Spider. Table 4 shows the results on Spider dataset. Since each question is standalone, we don't use interaction-level decoder or query editing. Our method can achieve the performance of 36.4% on dev set and 32.9% on test set, serving as a strong model for the context-independent cross-domain text-to-SQL generation. This demonstrates the effectiveness of our utterance-table encoder and table-aware decoder to handle the semantics of user utterances and the complexity of table schemas to generate complex SQL queries in unseen domains.

Furthermore, adding the utterance-table BERT

[4] More details at https://github.com/taoyds/spider/tree/master/evaluation_examples

[5] https://github.com/google-research/bert

	Question Match		Interaction Match	
	Dev	Test	Dev	Test
SyntaxSQL-con (Yu et al., 2019b)	18.5	20.2	4.3	5.2
CD-Seq2Seq (Yu et al., 2019b)*	21.9	23.2	8.1	7.5
+ segment copy (w/ predicted query)	21.7	20.3	9.5	8.1
+ segment copy (w/ gold query)	27.3	26.7	10.0	8.4
Ours	31.4	–	14.7	–
+ query attention and sequence editing (w/ predicted query)	33.0	–	16.4	–
+ query attention and sequence editing (w/ gold query)	40.6	–	17.3	–
Ours + utterance-table BERT Embedding	40.4	–	18.1	–
+ query attention (w/ predicted query)	42.7	–	21.6	–
+ query attention and sequence editing (w/ predicted query)	**47.2**	**47.9**	**29.5**	**25.3**
+ query attention and sequence editing (w/ gold query)	53.4	54.5	29.2	25.0

Table 5: SParC results. For our models, we only report test set results of our best model on the dev set. *We improve the CD-Seq2Seq performance over Yu et al. (2019b) by separating and parsing the column names (e.g., stu_fname → student first name) and using the schema-specific output vocabulary during decoding.

	Dev Set			Test Set		
	Query	Relaxed	Strict	Query	Relaxed	Strict
FULL (Suhr et al., 2018)	$37.5_{\pm 0.9}$	$63.0_{\pm 0.7}$	$62.5_{\pm 0.9}$	$43.6_{\pm 1.0}$	$69.3_{\pm 0.8}$	$69.2_{\pm 0.8}$
Ours	36.2	60.5	60.0	43.9	68.5	68.1

Table 6: ATIS results on dev set and test set.

embedding gives significant improvement, achieving 57.6% on dev set and 53.4% on test set, which is comparable to the state-of-the-art results from IRNet with BERT. We attribute our BERT model's high performance to (1) the empirically powerful text understanding ability of pretrained BERT model and (2) the early interaction between utterances and column headers when they are concatenated in a single sequence as the BERT input.

SParC. Table 5 shows the results on SParC dataset. Similar to Spider, our model without previous query as input already outperforms SyntaxSQL-con, achieving 31.4% question matching accuracy and 14.7% interaction matching accuracy. In addition, compared with CD-Seq2Seq, our model enjoys the benefits of the table-utterance encoder, turn attention, and the joint consideration of utterances and table schemas during the decoding stage. This boosts the performance by 10% question accuracy and 6% interaction accuracy.

Furthermore, we also investigate the effect of copying segment. We use the same segment copy procedure as Suhr et al. (2018): first deterministically extract segments from the previous query and encode each segment using an LSTM, then generate a segment by computing its output probability based on its segment encoding. However, since the segment extraction from Suhr et al. (2018) is exclusively designed for the ATIS dataset, we implement our own segment extraction procedure by extracting `SELECT`, `FROM`, `GROUP BY`, `ORDER BY` clauses as well as different conditions in `WHERE` clauses. In this way, 3.9 segments can be extracted per SQL on average. We found that adding segment copying to CD-Seq2Seq gives a slightly lower performance on question matching and a small gain on interaction matching, while using segments extracted from the gold query can have much higher results. This demonstrates that segment copy is vulnerable to error propagation. In addition, it can only copy whole segments hence has difficulty capturing the changes of only one or a few tokens in the query.

To better understand how models perform as the interaction proceeds, Figure 4 (Left) shows the performance split by turns on the dev set. The questions asked in later turns are more difficult to answer given longer context history. While the baselines have lower performance as the turn number increases, our model still maintains 38%-48% accuracy for turn 2 and 3, and 20% at turn 4 or beyond. Similarly, Figure 4 (Right) shows the performance split by hardness levels with the frequency of examples. This also demonstrates our model is more competitive in answering hard and extra hard questions.

ATIS. We also report our model performance

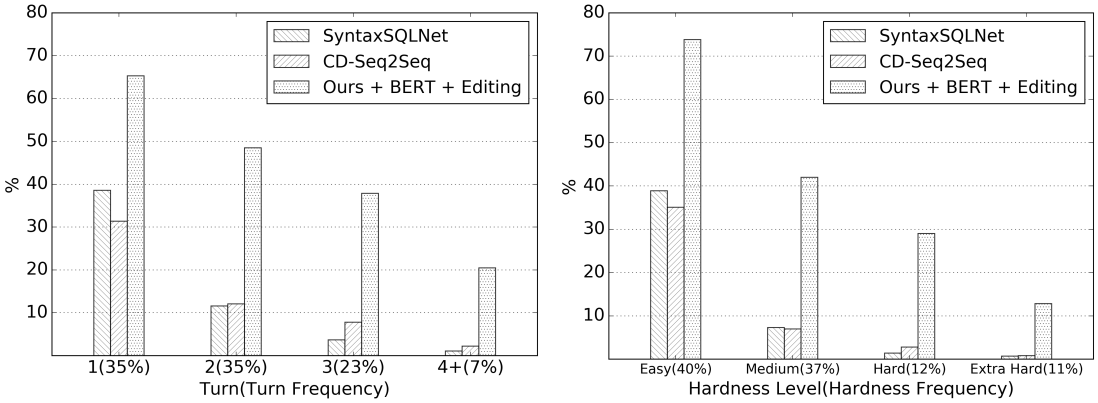


Figure 4: Performance split by different turns (Left) and hardness levels (Right) on SParC dev set.

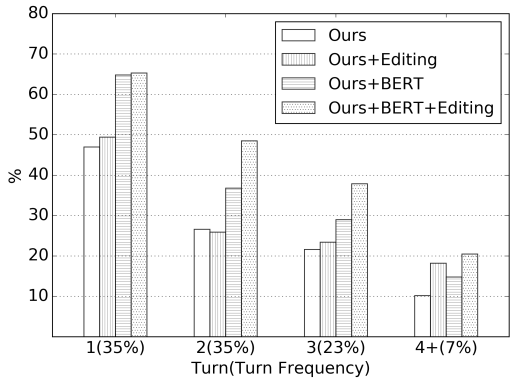


Figure 5: Effect of query editing at different turns on SParC dev set.

on ATIS in Table 6. Our model achieves 36.2% dev and 43.9% test string accuracy, comparable to Suhr et al. (2018). On ATIS, we only apply our editing mechanism and reuse their utterance encoder instead of the BERT utterance-table encoder, because ATIS is single domain.

5.5 Effect of Query Editing

We further investigate the effect of our query editing mechanism. To this end, we apply editing from both the gold query and the predicted query on our model with or without the utterance-table BERT embedding. We also perform an ablation study to validate the contribution of query attention and sequence editing separately.

As shown in Table 5, editing the gold query consistently improves both question match and interaction match accuracy. This shows the editing approach is indeed helpful to improve the generation quality when the previous query is the oracle.

Using the predicted query is a more realistic setting, and in this case, the model is affected by error propagation due to the incorrect queries produced by itself. For the model without the utterance-table BERT embedding, using the predicted query only gives around 1.5% improvement. As shown in Figure 5, this is because the editing mechanism is more helpful for turn 4 which is a small fraction of all question examples. For the model with the utterance-table BERT embedding, the query generation accuracy at each turn is significantly improved, thus reducing the error propagation effect. In this case, the editing approach delivers consistent improvements of 7% increase on question matching accuracy and 11% increase on interaction matching accuracy. Figure 5 also shows that query editing with BERT benefits all turns.

Finally, as an ablation study, Table 5 also reports the result with only query attention (use predicted query) on the dev set. This improves over our vanilla BERT model without query attention and achieves 42.7% question and 21.6% interaction matching accuracy. With query editing, our best model further improves to 47.2% question and 29.5% interaction matching accuracy. This demonstrates the effectiveness of our query attention and query editing separately, both of which are essential to make use of the previous query.

6 Conclusions

In this paper, we propose an editing-based encoder-decoder model to address the problem of context-dependent cross-domain text-to-SQL generation. While being simple, empirical results demonstrate the benefits of our editing mechanism. The approach is more robust to error propagation than copying segments, and its performance increases when the basic text-to-SQL generation quality (without editing) is better.

Acknowledgements

We thank the anonymous reviewers for their thoughtful detailed comments.

References

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the conference on empirical methods in natural language processing*, pages 421–432. Association for Computational Linguistics.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *ACL*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 18–27. Association for Computational Linguistics.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994a. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, Stroudsburg, PA, USA. Association for Computational Linguistics.

Deborah A Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994b. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the workshop on Human Language Technology*, pages 43–48. Association for Computational Linguistics.

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

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

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)*.

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)*.

Daniel Fried, Jacob Andreas, and Dan Klein. 2018. Unified pragmatic models for generating and following instructions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question generation from sql queries improves neural semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

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

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. Dialsql: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy S Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. In *Advances in Neural Information Processing Systems*, pages 10052–10062.

Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. 2018. Retrieval-based neural code generation. In *EMNLP*.

Charles T Hemphill, John J Godfrey, and George R Doddington. 1990. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Hsin-Yuan Huang, Eunsol Choi, and Wen-tau Yih. 2019. Flowqa: Grasping flow in history for conversational machine comprehension. In *ICLR*.

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)*.

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

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

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*.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Dongjun Lee. 2019. Recursive and clause-wise decoding for complex and cross-domain text-to-sql generation. In *CoRR*.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS 2017 Workshop Autodiff*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

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

Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. 2018. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *arXiv preprint arXiv:1809.05054*.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Alane Suhr and Yoav Artzi. 2018. Situated mapping of sequential instructions to actions with single-step reward observation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware sql generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Andreas Vlachos and Stephen Clark. 2014. A new corpus and imitation learning framework for context-dependent semantic parsing. *Transactions of the Association for Computational Linguistics*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

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*.

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)*.

Pengcheng Yin, Graham Neubig, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L Gaunt. 2019. Learning to represent edits. In *ICLR*.

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 NAACL*. Association for Computational Linguistics.

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*.

Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics.

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

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

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI*, pages 1050–1055, Portland, OR. AAAI Press/MIT Press.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *UAI*.

Luke S Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 976–984. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.