NL-EDIT: Correcting Semantic Parse Errors through Natural Language Interaction

Ahmed Elgohary[★][*], Christopher Meek[♣], Matthew Richardson[♣], Adam Fourney[♣],
Gonzalo Ramos[♣], Ahmed Hassan Awadallah[♣]
[★]University of Maryland [♣]Microsoft Research
elgohary@cs.umd.edu {meek,mattri,adamfo,goramos,hassanam}@microsoft.com

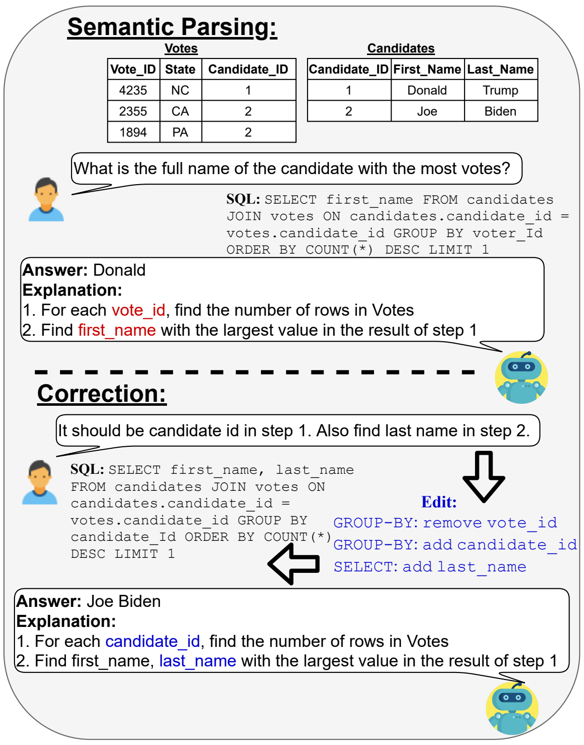
Abstract

We study semantic parsing in an interactive setting in which users correct errors with natural language feedback. We present NL-EDIT, a model for interpreting natural language feedback in the interaction context to generate a sequence of edits that can be applied to the initial parse to correct its errors. We show that NL-EDIT can boost the accuracy of existing text-to-SQL parsers by up to 20% with only one round of correction. We analyze the limitations of the model and discuss directions for improvement and evaluation. The code and datasets used in this paper are publicly available at http://aka.ms/NLEdit.

1 Introduction

Major progress in natural language processing has been made towards fully automating challenging tasks such as question answering, translation, and summarization. On the other hand, several studies have argued that machine learning systems that can explain their own predictions (Doshi-Velez and Kim, 2017) and learn interactively from their end-users (Amershi et al., 2014) can result in better user experiences and more effective learning systems. We develop NL-EDIT—an approach that employs both explanations and interaction in the context of semantic parsing.

Most existing systems frame semantic parsing as a one-shot translation from a natural language question to the corresponding logical form (e.g., SQL query) (Yu et al., 2018a; Guo et al., 2019; Wang et al., 2020, inter alia). A growing body of recent work demonstrates that semantic parsing systems can be improved by including users in the parsing loop—giving them the affordance to examine the parses, judge their correctness, and provide feedback accordingly. The feedback often comes in the form of a binary correct/incorrect



Votes

Vote_ID	State	Candidate_ID
4235	NC	1
2355	CA	2
1894	PA	2

Candidates

Candidate_ID	First_Name	Last_Name
1	Donald	Trump
2	Joe	Biden

What is the full name of the candidate with the most votes?

SQL: `SELECT first_name FROM candidates JOIN votes ON candidates.candidate_id = votes.candidate_id GROUP BY voter_Id ORDER BY COUNT(*) DESC LIMIT 1`

Answer: Donald
Explanation:
1. For each vote_id, find the number of rows in Votes
2. Find first_name with the largest value in the result of step 1

It should be candidate id in step 1. Also find last name in step 2.

SQL: `SELECT first_name, last_name FROM candidates JOIN votes ON candidates.candidate_id = votes.candidate_id GROUP BY candidate_Id ORDER BY COUNT(*) DESC LIMIT 1`

```
GROUP-BY: remove vote_id
GROUP-BY: add candidate_id
SELECT: add last_name
```

Answer: Joe Biden
Explanation:
1. For each candidate_id, find the number of rows in Votes
2. Find first_name, last_name with the largest value in the result of step 1

Figure 1: Example human interaction with NL-EDIT to *correct* an initial parse through natural language feedback. In the **Semantic Parsing Phase** (top), an off-the-shelf parser generates an initial SQL query and provides an answer paired with an *explanation* of the generated SQL. In the **Correction Phase** (bottom), the user reviews the explanation and provides *feedback* that describes how the explanation should be corrected. The system parses the feedback as a set of *edits* that are applied to the initial parse to generate a corrected SQL.

signal (Iyer et al., 2017), answers to a multiple-choice question posed by the system (Gur et al., 2018; Yao et al., 2019), or suggestions of edits that can be applied to the parse (Su et al., 2018).

Unlike other frameworks for interactive semantic parsing that typically expect users to judge the correctness of the execution result or induced logical form, Elgohary et al. (2020) introduced a framework for interactive text-to-SQL in which induced SQL queries are fully explained in natural lan-

[*]Most of the work was done while the first author was an intern at Microsoft Research.

guage to users, who in turn, can correct such parses through natural language feedback (Figure 1). They construct the SPLASH dataset and use it to evaluate baselines for the semantic parse correction with natural language feedback task they introduce.

We present a detailed analysis of the feedback and the differences between the initial (incorrect) and the correct parse. We argue that a correction model should be able to interpret the feedback in the context of other elements of the interaction (the original question, the schema, and the explanation of the initial parse). We observe from SPLASH that most feedback utterances tend to describe a few edits that the user desires to apply to the initial parse. As such, we pose the correction task as a semantic parsing problem that aims to convert natural language feedback to a sequence of edits that can be deterministically applied to the initial parse to correct it. We use the edit-based modeling framework to show that we can effectively generate synthetic data to pre-train the correction model leading to clear performance gains.

We make the following contributions: (1) We present a scheme for representing SQL query Edits that benefits both the modeling and the analysis of the correction task, (2) we present NL-EDIT, an edit-based model for interactive text-to-SQL with natural language feedback. We show that NL-EDIT outperforms baselines in (Elgohary et al., 2020) by more than 16 points, (3) We demonstrate that we can generate synthetic data through the edit-based framing and that the model can effectively use this data to improve its accuracy and (4) We present a detailed analysis of the model performance including studying the effect of different components, generalization to errors of state-of-the-art parsers, and outline directions for future research.

2 Background

In the task of text-to-SQL parsing, the objective is given a database schema (tables, columns, and primary-foreign key relations) and a natural language question, generate a SQL query that answers the question when executed against the database. Several recent text-to-SQL models have been introduced (Yu et al., 2018a; Zhang et al., 2019; Guo et al., 2019; Wang et al., 2020, inter alia) as a result of the availability of SPIDER (Yu et al., 2018b), a large dataset of schema, questions and gold parses spanning several databases in different domains.

The task of SQL parse correction with natural language feedback (Elgohary et al., 2020) aims to correct an erroneous parse based on natural language feedback collected from the user. Given a question, a database schema, an incorrect initial parse, natural language feedback on the initial parse, the task is to generate a corrected parse.

To study this problem, Elgohary et al. (2020) introduced the SPLASH dataset. SPLASH was created by showing annotators questions and a natural language explanation of incorrect parses and asking them to provide feedback, in natural language, to correct the parse. The dataset contained 9,314 question-feedback pairs. Like the SPIDER dataset, it was split into train-dev-test sets by database to encourage the models to generalize to new unseen databases. They contrast the task with conversational semantic parsing (Suhr et al., 2018; Yu et al., 2019b,a; Andreas et al., 2020) and show that the two tasks are distinct and are addressing different aspects of utilizing context. They establish several baseline models and show that the task is challenging for state-of-the-art semantic parsing models. We use these as baselines for this work.

3 SQL Edits

We define a scheme for representing the edits required to transform one SQL query to another. We use that scheme both in our model and analysis. Our goal is to balance the granularity of the edits—too fine-grained edits result in complex structures that are challenging for models to learn, and too coarse-grained edits result in less compact structures that are harder for models to generate.

We view a SQL query as a set of clauses (e.g, `SELECT`, `FROM`, `WHERE`), each clause has a sequence of arguments (Figure 2). We mirror the SQL clauses `SELECT`, `FROM`, `WHERE`, `GROUP-BY`, `ORDER-BY`, `HAVING`, and `LIMIT`. For subqueries, we define a clause `SUBS` whose arguments are recursively defined as sets of clauses. Subqueries can be linked to the main query in two ways: either through an `IEU` clause (mirrors SQL `INTERSECT/EXCEPT/UNION`) whose first argument is one of the keywords `INTERSECT`, `EXCEPT`, `UNION` and its second argument is a pointer to a subquery in `SUBS`. The second is through nested queries where the arguments of some of the clauses (e.g., `WHERE`) can point at subqueries in `SUBS` (e.g., "`id NOT IN SUBS`$_1$").

With such view of two queries $\mathcal{P}_{source}$ and $\mathcal{P}_{target}$, we define their edit $\mathcal{D}_{source \rightarrow target}$ as

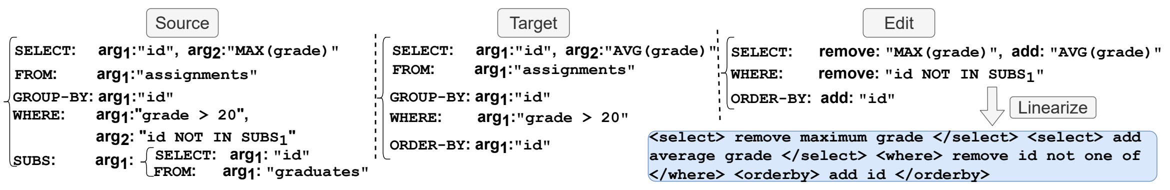

Source
SELECT: arg1:"id", arg2:"MAX(grade)"
FROM: arg1:"assignments"
GROUP-BY: arg1:"id"
WHERE: arg1:"grade > 20", arg2: "id NOT IN SUBS1"
SUBS: arg1: SELECT: arg1: "id" FROM: arg1: "graduates"
Target
SELECT: arg1:"id", arg2:"AVG(grade)"
FROM: arg1:"assignments"
GROUP-BY: arg1:"id"
WHERE: arg1:"grade > 20"
ORDER-BY: arg1:"id"
Edit
SELECT: remove: "MAX(grade)", add: "AVG(grade)"
WHERE: remove: "id NOT IN SUBS1"
ORDER-BY: add: "id"
Linearize
<select> remove maximum grade </select> <select> add average grade </select> <where> remove id not one of </where> <orderby> add id </orderby>

Figure 2: Edit for transforming the **source** query "`SELECT id, MAX(grade) FROM assignments WHERE grade > 20 AND id NOT IN (SELECT id from graduates) GROUP BY id`" to the **target** "`SELECT id, AVG(grade) FROM assignment WHERE grade > 20 GROUP BY id ORDER BY id`". The source and target are represented as sets of clauses (left and middle). The set of **edits** and its **linearized** form (Section 4) are shown on the right. Removing the condition "`id NOT IN SUBS`$_1$" makes the subquery unreferenced, hence pruned from the edit.

the set of clause-level edits $\{\mathcal{D}^{c}_{source \rightarrow target}\}$ for all types of clauses c that appear in $\mathcal{P}_{source}$ or $\mathcal{P}_{target}$ (Figure 2). To compare two clauses of type c, we simply exact-match their arguments: unmatched arguments in the source (e.g., `MAX(grade)` in `SELECT`) are added as to-remove arguments to the corresponding edit clause, and unmatched arguments in the target (e.g., "`id`" in the `ORDER-BY`) are added as to-add arguments.

Our current implementation follows SPIDER's assumption that the number of subqueries is at most one which implies that computing edits for different clauses can be done independently even for the clauses that reference a subquery (e.g., `WHERE` in Figure 2). The edit of the `SUBS` clause is recursively computed as the edit between two queries (any of them can be empty); the subquery of source and the subquery of target, i.e., $\mathcal{D}^{\texttt{SUBS}}_{source \rightarrow target} = \mathcal{D}_{source:\texttt{SUBS}_1 \rightarrow target:\texttt{SUBS}_1}$. We keep track of the edits to the arguments that reference the subquery. After all edit clauses are computed, we prune the edits of the `SUBS` clause if the subquery will no longer be referenced (`SUBS`$_1$ in Figure 2). We follow the SPIDER evaluation and discard the values in `WHERE/HAVING` clauses.

Throughout this paper, we refer to the number of add/remove operations in an edit as the **Edit Size**, and we denote it as $|\mathcal{D}_{source \rightarrow target}|$. For example, the edit in Figure 2 is of size four.

4 Model

We follow the task description in Section 2: the inputs to the model are the elements of the interaction—question, schema, an initial parse $\tilde{\mathcal{P}}$, and feedback. The model predicts a corrected $\bar{\mathcal{P}}$. The gold parse $\hat{\mathcal{P}}$ is available for training. Our model is based on integrating two key ideas in an encoder-decoder architecture. We start with a discussion of the intuitions behind the two ideas followed by the model details.

4.1 Intuitions

Interpreting feedback in context: The feedback is expected to link to all the other elements of the interaction (Figure 1). The feedback is provided in the context of the *explanation* of the initial parse, as a proxy to the parse itself. As such, the feedback tends to use the same terminology as the explanation. For example, the SQL explanations of (Elgohary et al., 2020) express "group by" in simple language "for each vote_id, find ...". As a result, human-provided feedback never uses "group by". We also notice that in several SPLASH examples, the feedback refers to particular steps in the explanation as in the examples in Figure 1. Unlike existing models (Elgohary et al., 2020), we replace the initial parse with its natural language explanation. Additionally, the feedback usually refers to columns/tables in the schema, and could often be ambiguous when examined in isolation. Such ambiguities can be usually resolved by relying on the context provided by the question. For example, "find last name" in Figure 1 is interpreted as "find last name besides first name" rather than "replace first name with last name" because the question asks for the "full name". Our first key idea is based on grounding the elements of the interaction by combining self-learned relations by transformer models (Vaswani et al., 2017) and hard-coded relations that we define according to the possible ways different elements can link to each other.

Feedback describes a set of edits: The difference between the erroneous parse and the correct one can mostly be described as a few edits that need to be applied to the initial parse to correct its errors (Section 7). Also, the feedback often only describes the edits to be made (Elgohary et al., 2020). As such, we can pose the task of correction with NL feedback as a semantic parsing task where we convert a natural language deception of

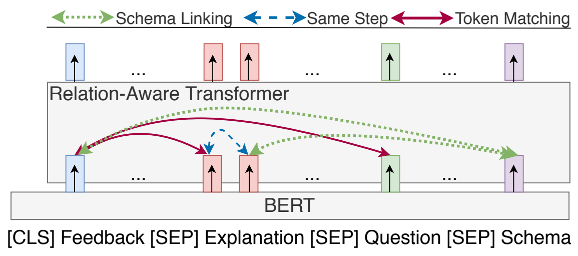

Schema Linking
Same Step
Token Matching
Relation-Aware Transformer
BERT
[CLS] Feedback [SEP] Explanation [SEP] Question [SEP] Schema

Figure 3: The Encoder of NL-EDIT grounds the feedback into the explanation, the question, and the schema by (1) passing the concatenation of their tokens through BERT, then (2) combining self-learned and hard-coded relations in a relation-aware transformer. Three types of relations (Interaction Relations) link the individual tokens of the inputs. Question-Schema and Schema-Schema relations are not shown.

the edits to a canonical form that can be applied deterministically to the initial parse to generate the corrected one. We train our model to generate SQL Edits (Section 3) rather than SQL queries.

4.2 Encoder

Our encoder (Figure 3) starts with passing the concatenation of the feedback, explanation, question, and schema through BERT (Devlin et al., 2019). Following (Wang et al., 2020; Suhr et al., 2018; Scholak et al., 2020), we tokenize the column/table names and concatenate them in one sequence (Schema) starting with the tokens of the tables followed by the tokens of the columns. Then, we average the BERT embeddings of the tokens corresponding to each column (table) to obtain one representation for the column (table).

Wang et al. (2020) study the text-to-SQL problem using the SPIDER dataset and show the benefit of injecting preexisting relations within the schema (column exists in a table, primary-foreign key), and between the question and schema items (column and table names) by: (1) name linking: link a question token to a column/table if the token and the item name match and (2) value linking: link a question token to a column if the token appears as a value under that column. To incorporate such relations in their model, they use the relation-aware self-attention formulation presented in (Shaw et al., 2018). The relation-aware transformer (Shaw et al., 2018) assigns a learned embedding for each relation type and combines such embeddings with the self-attention of the original transformer model (Vaswani et al., 2017): If a preexisting relation r holds between two tokens, the embedding of r is added as a bias term to the self-attention computation between the two tokens.

In addition to those relations, we define a new set of relations that aim at contextualizing the feedback with respect to the other elements of the interaction in our setup: (1) ***[Feedback-Schema]*** We link the feedback to the schema the same way the question is linked to the schema via both name and value linking, (2) ***[Explanation-Schema]*** Columns and tables are mentioned with their exact names in the explanation. We link the explanation to the schema only through exact name matching, (3) ***[Feedback-Question]*** We use partial (at the lemma level) and exact matching to link tokens in the feedback and the question, (4) ***[Feedback-Explanation]*** We link tokens in the feedback to tokens in the explanation through partial and exact token matching. Since the feedback often refers to particular steps, we link the feedback tokens to explanation tokens that occur in steps that are referred to in the feedback with a separate relation type that indicates step reference in the feedback, and (5) ***[Explanation-Explanation]*** We link explanation tokens that occur within the same step. We use the same formulation of relation-aware self-attention as (Wang et al., 2020) and add the relation-aware layers on top of BERT to integrate all relations into the model (Figure 3).

4.3 Decoder

Using a standard teacher-forced cross-entropy loss, we train our model to generate *linearized* SQL Edits (Figure 2). At training time, we compute the reference SQL Edit $\mathcal{D}_{\tilde{\mathcal{P}}\to\hat{\mathcal{P}}}$ of the initial parse $\tilde{\mathcal{P}}$ and the gold parse $\hat{\mathcal{P}}$ (Section 3). Then we linearize $\mathcal{D}_{\tilde{\mathcal{P}}\to\hat{\mathcal{P}}}$ by listing the clause edits in a fixed order (`FROM`, `WHERE`, `GROUP-BY`, ... etc.). The argument of each clause—representing one add or remove operation—is formatted as `<CLAUSE> ADD/REMOVE ARG </CLAUSE>`. We express SQL operators in `ARG` with natural language explanation as in (Elgohary et al., 2020). For example, the argument "`AVG(grade)`" is expressed as "average grade". At inference time, we generate a corrected parse $\bar{\mathcal{P}}$ by applying the produced edit to the initial parse $\tilde{\mathcal{P}}$.

We use a standard transformer decoder that either generates tokens from the output vocab or copies columns and tables from the encoder output. Since all editing operations should be directed by the feedback, we tried splitting the attention to the encoder into two phases: First, we attend to the feedback only and update the decoder state

Replace-Select-Column:
- replace {NEW-COL} with {OLD-COL}
- you should find {OLD-COL} instead
Add-Where-Condition:
- delete {COL} {OPERATOR} {VALUE}
Remove-Limit:
- only top {LIMIT-VALUE} rows are needed

Table 1: Example SQL Editors with corresponding feedback templates. The synthesized feedback is reversing the edit applied to a correct SQL as our synthesis process starts with the gold SQL and reaches an initial SQL after applying the edit.

accordingly. Then, we use the updated decoder state to attend to the other inputs. With that, we only observed a marginal improvement of 0.5% in the accuracy. We conduct all our experiments with standard decoder-encoder attention and plan to investigate other attention patterns in the future.

5 Synthetic Feedback

In this section, we describe our process for automatically synthesizing additional examples for training the correction model. Recall that each example consists of a question about a given schema paired with a gold parse, an initial erroneous parse, and feedback. Starting with a seed of questions and their corresponding gold parses from SPIDER's training set (8,099 pairs)[1], our synthesis process applies a sequence of SQL editing operations to the gold parse to reach an altered parse that we use as the initial parse (Algorithm 1).

By manually inspecting the edits (Section 3) we induce for the initial and gold parses in SPLASH training set, we define 26 SQL editors and pair each editor with their most frequent corresponding feedback template(s) (Examples in Table 1). We also associate each editor with a set of constraints that determines whether it can be applied to a given SQL query (e.g., the "Remove-Limit" editor can only be applied to a query that has a limit clause).

Algorithm 1 summarizes the synthesis process. We start by creating N (controls the size of the dataset) clones of each seed example. Elgohary et al. (2020)'s analysis of SPLASH shows that multiple mistakes might be present in the initial SQL, hence we allow our synthesis process to introduce up to four edits (randomly decided in line:4) to each clone p. For each editing step, we sample a feasible edit for the current parse (line:5) with man-

Algorithm 1 Training Data Synthesis

```
1: for seed in SPIDER training set do
2:     for $p$ in CLONE(seed, $N$) do
3:         feedback = []
4:         for $i = 1$ : RAND-NUM-EDITS() do
5:             $e \leftarrow$ RAND-FEASIBLE-EDIT($p$)
6:             $p$.APPLY-EDIT($e$)
7:             feedback.ADD($e$.FEEDBACK())
8:         output: seed.DB, seed.Question, $p$,
9:                 feedback, seed.Gold-SQL
```

ually set probabilities for each edit to balance the number of times each editor is applied in the final dataset. Applying an edit (line:6) involves sampling columns/tables from the current parse and/or the schema, sampling operators and values for altering conditions, and populating the corresponding feedback template. We combine the feedback of all the applied editors into one string and use it as the feedback of the synthesized example.

6 Experiments

Setup: We conduct our experiments using SPLASH (Elgohary et al., 2020) (Section 2) whose train, dev, and test sets are of sizes 7481, 871, and 962, respectively. Using our feedback synthesis process (Section 5), we generate 50,000 additional synthetic training examples. In our preliminary experiments, We found that training the model on the synthetic dataset first then continuing on SPLASH outperforms mixing the synthetic and real examples and training on both of them simultaneously. We train the model on the synthetic examples for 20,000 steps and continue training on the real examples until reaching 100,000 steps in total. We choose the best checkpoint based on the development set accuracy. We varied the number of training steps on the synthetic examples and 20,000 steps achieved the highest accuracy on the dev set.

We use BERT-base-uncased (Devlin et al., 2019) in all our experiments. We set the number of layers in the relational-aware transformer to eight (Wang et al., 2020) and the number of decoder layers to two. We train with batches of size 24. We use the Adam optimizer (Kingma and Ba, 2015) for training. We freeze BERT parameters during the first 5,000 warm-up steps and update the rest of the parameters with a linearly increasing learning rate from zero to 5×10^{-4}. Then, we linearly decrease the learning rates from 5×10^{-5} for BERT and

[1]We ensure there is no overlap between examples in the seed and the dev set of SPLASH.

	Correction Acc. (%)	Edit ↓ (%)	Edit ↑ (%)	Progress (%)
Rule-based Re-ranking	16.63	38.35	32.81	-15.67
EditSQL+Feedback	25.16	47.44	23.51	7.71
NL-EDIT (Ours)	**41.17**	**72.41**	**16.93**	**36.99**
Oracle Re-ranking	36.38	34.69	1.04	31.22

Table 2: Comparing NL-EDIT to **baselines** in (Elgohary et al., 2020): Rule-based Re-ranking and EditSQL+Feedback and to the beam re-ranking upper-bound. **Edit** ↓ (**Edit** ↑) is the percentage of examples on which the number of edits/errors strictly decreased (increased). **Progress** is the average relative reduction in the number of edits (Section 6). Elgohary et al. (2020) estimate the upper-bound on the correction accuracy as 81.5%.

5×10^{-4} for the other parameters to zero.[2] We use beam search with a beam of size 20 and take the top-ranked beam that results in a valid SQL after applying the inferred edit.

Evaluation: We follow (Elgohary et al., 2020) and use the *correction accuracy* as our main evaluation measure: each example in SPLASH test set contains an initial parse $\tilde{\mathcal{P}}$ and a gold parse $\hat{\mathcal{P}}$. With a predicted (corrected) parse by a correction model $\bar{\mathcal{P}}$, they compute the correction accuracy using the exact-set-match (Yu et al., 2018b) between $\bar{\mathcal{P}}$ and $\hat{\mathcal{P}}$ averaged over all test examples. While useful, correction accuracy also has limitations. It expects models to be able to fully correct an erroneous parse with only one utterance of feedback as such, it is defined in terms of the exact match between the corrected and the gold parse. We find (Table 2) that in several cases, models were still able to make *progress* by reducing the number of errors as measured by the edit size (Section 3) after correction. As such, we define another set of metrics to measure partial progress. We report (Edit ↓ and Edit ↑ in Table 2) the percentage of examples on which the size of the edit set strictly decreased/increased. To combine Edit ↓ and Edit ↑ in one measure and account for the relative reduction (increase) in the number of edits, we define

$$\text{Progress}(\mathcal{S}) = \frac{1}{|S|} \sum_{\tilde{\mathcal{P}},\bar{\mathcal{P}},\hat{\mathcal{P}} \in \mathcal{S}} \frac{|\mathcal{D}_{\tilde{\mathcal{P}}\to\hat{\mathcal{P}}}| - |\mathcal{D}_{\bar{\mathcal{P}}\to\hat{\mathcal{P}}}|}{|\mathcal{D}_{\tilde{\mathcal{P}}\to\hat{\mathcal{P}}}|}.$$

Given a test set $\mathcal{S}$, the Progress of a correction model is computed as the average relative edit reduction between the initial parse $\tilde{\mathcal{P}}$ and the gold parse $\hat{\mathcal{P}}$ by predicting a correction $\bar{\mathcal{P}}$ of $\tilde{\mathcal{P}}$. A perfect model that can fully correct all errors in the initial parse would achieve a 100% progress. A model can have a negative progress (e.g., Rule-based re-ranking in Table 2) when it frequently predicts corrections with more errors than those in the initial parse. Unlike correction accuracy, Progress is more aligned with user experience in an interactive environment (Su et al., 2018) as it assigns partial credit for fixing a subset of the errors and also, it penalizes models that predict an even more erroneous parse after receiving feedback.

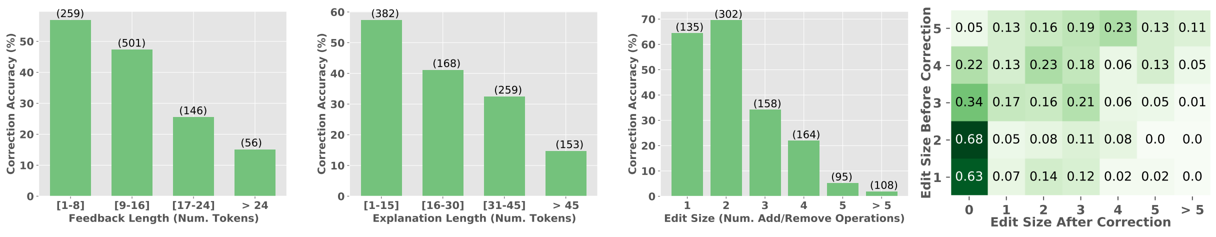
Results: We compare (Table 2) NL-EDIT to the two top-performing baselines in (Elgohary et al., 2020) and also to the beam re-ranking upper-bound they report. NL-EDIT significantly increases the correction accuracy over the top baseline (EditSQL+Feedback) by more than 16% and it also outperforms oracle re-ranking by around 5%. We also note that in 72.4% of the test examples, NL-EDIT was able to strictly reduce the number of errors in the initial parse (Edit ↓) which potentially indicates a more positive user experience than the other models. NL-EDIT achieves 37% Progress which indicates faster convergence to the fully corrected parse than all the other models.

7 Analysis

7.1 Ablations

Following the same experimental setup in Section 6, we compare NL-EDIT to other variants with one ablated component at a time (Table 3). We ablate the **feedback**, the **explanation**, and the **question** from the encoder input. We also ablate the **interaction relations** (Section 4.2) that we incorporate in the relation-aware transformer module. We only ablate the new relations we introduce to model the interaction (shown in Figure 3), but we keep the Question-Schema and Schema-Schema relations introduced in (Wang et al., 2020). For each such variant, we train for 20,000 steps on the synthetic dataset then continue training on SPLASH until step 100,000. We also train an ablated variant that does not use the **synthetic feedback** where we

[2]The learning rate schedule is only dependent on the step number regardless of whether we are training on the synthetic data or SPLASH. We tried resetting the learning rates back to their maximum values after switching to SPLASH, but did not observe any improvement in accuracy.



(a) Feedback Length (b) Explanation Length (c) Reference Edit Size (d) Transition of Edit Size

Figure 4: **a-c**: Breakdown of the correction accuracy on SPLASH test set by **(a)** feedback length, **(b)** explanation length, and **(c)** size of the reference edit (number of add or remove operations). The number of examples in each group is shown on top of the bars. **d:** Transitions in edit size after correction. For each edit size of the initial parse (rows), we show the distribution of the edit size after correction.

NL-EDIT	41.17
– Feedback	19.81
– Explanation	26.80
– Question	38.27
– Interaction Relations	35.35
– Synthetic Feedback	35.01

Table 3: Correction accuracy on SPLASH Test of NL-EDIT versus variants with one ablated component each.

Long Feedback Not Describing an Edit:
"you should determine the major record format from the orchestra table and make sure it is arranged in ascending order of number of rows that appear for each major record format."
Long Feedback Describing an Edit:
"replace course id (both) with degree program id, first courses with student enrolment, course description with degree summary name, second courses with degree programs."

Table 4: Example long feedback that NL-EDIT struggles with. Top: The feedback describes a rewriting of the query rather than how to edit it. Bottom: The initial query has several errors and the feedback enumerates how to edit all of them.

train for 100,000 steps only on SPLASH. For all variants, we choose the checkpoint with the largest correction accuracy on the dev set and report the accuracy on the SPLASH test set.

The results in Table 3 confirm the effectiveness of each component in our model. We find that the model is able to correct 19.8% of the examples without the feedback. We noticed that the ablated-feedback model almost reaches that accuracy only after training on the synthetic data with very minor improvement (< 1%) after training on SPLASH. Only using the question and the explanation, the model is able to learn about a set of systematic errors that parsers make and how they can be corrected (Gupta et al., 2017; Yin and Neubig, 2019).

7.2 Error Analysis

In Figure 4, we breakdown the correction accuracy by the feedback and explanation lengths (in number of tokens) and by the reference edit size (number of required edit operations to fully correct the initial parse). The accuracy drops significantly when the reference edit size exceeds two (Figure 4c), while it declines more gradually as the feedback and explanation increase in length. We manually (Examples in Table 4) inspected the examples with longer feedback than 24, and found that 8% of them the feedback is long because it describes how to rewrite the whole query rather than being limited to only the edits to be made. In the remaining 92%, the initial query had several errors (edit size of 5.5 on average) with the corresponding feedback enumerating all of them.

Figure 4d shows how the number of errors (measured in edit size) changes after correction. The figure shows that even for examples with a large number of errors (four and five), the model is still able to reduce the number of errors in most cases. We manually inspected the examples with only one error that the model failed to correct. We found 15% of them have either wrong or non-editing feedback and in 29% the model produced the correct edit but with additional irrelevant ones. The dominant source of error in the remaining examples is because of failures with linking the feedback to the schema (Examples in Table 5).

7.3 Cross-Parser Generalization

So far, we have been using SPLASH for both training and testing. The erroneous parses (and corresponding feedback) in SPLASH are based on the Seq2Struct parser (Shin, 2019). Recent progress

Adding extra edits:
Ques.: Which city and country is the Alton airport at? **Initial:** `SELECT City, Country FROM airports WHERE AirportName = 'Alton' AND Country = 'USA'` **Feedback:** remove "and country equals USA" phrase. **Predicted:** `<where> remove AirportName equals </where> <where> remove Country equals </where>` **Gold:** `<where> remove AirportName equals </where>`
Failing to link feedback and schema:
Ques.: What are the full names of all left handed players, in order of birth date? **Initial:** `SELECT first_name, last_name FROM players ORDER BY birth_date Asc` **Feedback:** make sure that player are left handed. **Predicted:** `<where> add birth_date equals </where>` **Gold:** `<where> add hand equals </where>`

Table 5: Example failure cases of NL-EDIT.

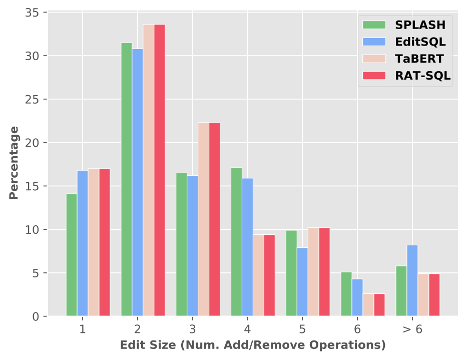


Figure 5: Distribution of Edit Size per example in SPLASH compared to the generalization test sets constructed based on EditSQL, TaBERT, and RAT-SQL.

in model architectures (Wang et al., 2020) and pretraining (Yin et al., 2020; Yu et al., 2021a) has led to parsers that already outperform Seq2Struct by more than 30% in parsing accuracy.[3] Here, we ask whether NL-EDIT that we train on SPLASH (and synthetic feedback) can generalize to parsing errors made by more recent parsers without additional parser-specific training data.

We follow the same crowdsourcing process used to construct SPLASH (Section 2) to collect three new test sets based on three recent text-to-SQL parsers: EditSQL (Zhang et al., 2019), TaBERT (Yin et al., 2020) and RAT-SQL (Wang et al., 2020). Following Elgohary et al. (2020), we run each parser on SPIDER dev set and only collect feedback for the examples with incorrect parses that can be explained using their SQL explanation framework. Table 6 (Top) summarizes the three new test sets and compares them to SPLASH test set. We note that the four datasets are based on the same set of questions and databases (SPIDER dev).

Table 6 (Bottom) compares the parsing accuracy (measure by exact query match (Yu et al., 2018b)) of each parser when used by itself **(No Interaction)** to integrating it with NL-EDIT. We report both the accuracy on the examples provided to NL-EDIT (**Error Correction**) and the **End-to-End** accuracy on the full SPIDER dev set. NL-EDIT significantly boosts the accuracy of all parsers, but with a notable drop in the gains as the accuracy of the parser improves. To explain that, in Figure 5 we compare the distribution of reference edit size across the four datasets. The figure does not show any significant differences in the distributions that would lead to such a drop in accuracy gain. Likewise, the distributions of the feedback lengths are very similar (the mean is shown in Table 6). As parsers improve in accuracy, they tend to make most of their errors on complex SQL queries. Although the number of errors with each query does not significantly change (Figure 5), we hypothesize that localizing the errors in a complex initial parse, with a long explanation (Table 6), is the main generalization bottleneck that future work needs to address.

8 Related Work and Discussion

Natural language to SQL: Natural language interfaces to databases have been an active field of study for many years (Woods et al., 1972; Warren and Pereira, 1982; Popescu et al., 2003; Li and Jagadish, 2014). The development of new large scale datasets, such as WikiSQL (Zhong et al., 2017) and SPIDER (Yu et al., 2018b), has reignited the interest in this area with several new models introduced recently (Choi et al., 2020; Wang et al., 2020; Scholak et al., 2020). Another related line of work has focused on conversation semantic parsing, e.g. SParC (Yu et al., 2019b), CoSQL (Yu et al., 2019a), and SMCalFlow (Andreas et al., 2020), where parsers aim at modeling utterance sequentially and in context of previous utterances.

Interactive Semantic Parsing: Several previous studies have looked at the problem of improving semantic parser with feedback or human interactions (Clarke et al., 2010; Artzi and Zettlemoyer, 2013). Interactions are supported in multiple ways including binary correct/incorrect signal (Iyer et al., 2017), answers to a yes/no or a multiple-choice

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

	Seq2Struct (SPLASH)	EditSQL	TaBERT	RAT-SQL
Correction Test Sets Summary				
Number of Examples	962	330	267	208
Average Feedback Length	13.1	13.5	12.9	12.2
Average Explanation Length	26.4	28.3	32.2.9	34.0
Semantic Parsing Accuracy (%)				
Error Correction	41.1	28.0	22.7	21.3
No Interaction	<u>41.3</u>	<u>57.6</u>	<u>65.2</u>	<u>69.7</u>
End-to-End	<u>61.6</u>	<u>66.6</u>	<u>71.1</u>	<u>74.0</u>
Δ w/ Interaction	**+20.3**	**+8.9**	**+5.9**	**+4.3**

Table 6: Evaluating the zero-shot generalization of NL-EDIT to different parsers (EditSQL, TaBERT, and RAT-SQL) after training on SPLASH that is constructed based on the Seq2Struct parser. Top: Summary of the dataset constructed based on each parser. Feedback and explanation length is the number of tokens. Bottom: The **Error Correction** accuracy on each test set and the end-to-end accuracy of each parser <u>on the full SPIDER dev set</u> with and without interaction. **Δ w/ Interaction** is the gain in end-to-end accuracy with the interaction added.

question posed by the system (Yao et al., 2019; Gur et al., 2018) or suggestions of edits that can be applied to the parse (Su et al., 2018).

Yao et al. (2019) and Gur et al. (2018) ask yes/no and multiple-choice questions and use the answers in generating the pars. Elgohary et al. (2020) introduce SPLASH (Section 2), a dataset for correcting semantic parsing with natural language feedback. Using language as a medium for providing feedback enables the human to provide rich open-form feedback in their natural way of communication giving them control and flexibility specifying what is wrong and how it should be corrected. Our work uses SPLASH and proposes to pose the problem of semantic parse correction as a parser editing problem with natural language feedback input. This is also related to recent work on casting text generation (e.g. summarization, grammatical error correction, sentence splitting, etc.) as a text editing task (Malmi et al., 2019; Panthaplackel et al., 2020; Stahlberg and Kumar, 2020) where target texts are reconstructed from inputs using several edit operations.

Semantic Parsing with Synthetic Data: Semantic parsing systems have frequently used synthesized data to alleviate the challenge of labeled data scarcity. In their semantic parser overnight work, Wang et al. (2015) proposed a method for training semantic parsers quickly in a new domain using synthetic data. They generate logical forms and canonical utterances and then paraphrase the canonical utterances via crowd-sourcing. Several other approaches have demonstrated the benefit of adopting this approach to train semantic parsers in low-resource settings (Su et al., 2017; Zhong et al., 2017; Cheng et al., 2018; Xu et al., 2020). Most recently, synthetic data was used to continue to pre-train language models for semantic parsing tasks (Herzig et al., 2020; Yu et al., 2021a,b). We build on this line work by showing that we can generate synthetic data automatically without human involvement to simulate edits between an erroneous parse and a correct one.

9 Conclusions and Future Work

We introduced a model, a data augmentation method, and analysis tools for correcting semantic parse errors in text-to-SQL through natural language feedback. Compared to previous models, our model improves the correction accuracy by 16% and boosts the end-to-end parsing accuracy by up to 20% with only one turn of feedback. Our work creates several avenues for future work: (1) improving the model by better modeling the interaction between the inputs and exploring different patterns for decoder-encoder attention, (2) evaluating existing methods for training with synthetic data (e.g., curriculum learning (Bengio et al., 2009)), (3) optimizing the correction model for better user experience using the progress measure we introduce, and (4) using the SQL edits scheme in other related tasks such as conversational text-to-SQL parsing.

Acknowledgments

This work has benefited greatly from discussions with Xiang Deng, Alex Polozov, Tao Yu, and Guoqing Zheng. We thank Pengcheng Yin for sharing TaBERT predictions before the official code release. We are very grateful to our reviewers for their insightful feedback and suggestions.

References

Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35.

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the International Conference of Machine Learning*.

Jianpeng Cheng, Siva Reddy, and Mirella Lapata. 2018. Building a neural semantic parser from a domain ontology. *ArXiv*, abs/1812.10037.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. *arXiv preprint arXiv:2004.03125*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Conference on Computational Natural Language Learning*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your parser: Interactive text-to-SQL with natural language feedback. In *Proceedings of the Association for Computational Linguistics*.

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 *Proceedings of the Association for Computational Linguistics*.

Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. Deepfix: Fixing common c language errors by deep learning. In *Association for the Advancement of Artificial Intelligence*.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue based structured query generation. In *Proceedings of the Association for Computational Linguistics*.

Jonathan Herzig, P. Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TAPAS: Weakly supervised table parsing via pre-training. In *Proceedings of the Association for Computational Linguistics*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the Association for Computational Linguistics*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. In *Proceedings of the VLDB Endowment*.

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. 2019. Encode, tag, realize: High-precision text editing. In *Proceedings of Empirical Methods in Natural Language Processing*.

Sheena Panthaplackel, Pengyu Nie, Milos Gligoric, Junyi Jessy Li, and Raymond Mooney. 2020. Learning to update natural language comments based on code changes. In *Proceedings of the Association for Computational Linguistics*.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *International Conference on Intelligent User Interfaces*.

Torsten Scholak, Raymond Li, Dzmitry Bahdanau, Harm de Vries, and Chris Pal. 2020. DuoRAT: Towards simpler text-to-SQL models. *ArXiv preprint arXiv:2010.11119*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Richard Shin. 2019. Encoding database schemas with relation-aware self-attention for text-to-SQL parsers. *arXiv preprint arXiv:1906.11790*.

Felix Stahlberg and Shankar Kumar. 2020. Seq2Edits: Sequence transduction using span-level edit operations. In *Proceedings of Empirical Methods in Natural Language Processing*.

Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, P. Pantel, M. Gamon, and Mark J. Encarnación. 2017. Building natural language interfaces to web APIs. In *Proceedings of the ACM International Conference on Information and Knowledge Management*.

Yu Su, Ahmed Hassan Awadallah, Miaosen Wang, and Ryen W White. 2018. Natural language interfaces with fine-grained user interaction: A case study on web APIs. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

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 *Proceedings of Advances in Neural Information Processing Systems*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the Association for Computational Linguistics*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the Association for Computational Linguistics*.

David H.D. Warren and Fernando C.N. Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8.

W. A. Woods, Ronald M Kaplan, and Bonnie L. Webber. 1972. The lunar sciences natural language information system: Final report. *BBN Report 2378*.

Silei Xu, Sina Semnani, Giovanni Campagna, and Monica Lam. 2020. AutoQA: From databases to QA semantic parsers with only synthetic training data. In *Proceedings of Empirical Methods in Natural Language Processing*.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Proceedings of Empirical Methods in Natural Language Processing*.

Pengcheng Yin and Graham Neubig. 2019. Reranking for neural semantic parsing. In *Proceedings of the Association for Computational Linguistics*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the Association for Computational Linguistics*.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2021a. GraPPa: Grammar-augmented pre-training for table semantic parsing. In *Proceedings of the International Conference on Learning Representations*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018a. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of Empirical Methods in Natural Language Processing*.

Tao Yu, Rui Zhang, Heyang 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 Empirical Methods in Natural Language Processing*.

Tao Yu, Rui Zhang, Alex Polozov, Christopher Meek, and Ahmed Hassan Awadallah. 2021b. SCoRe: Pre-training for context representation in conversational semantic parsing. In *Proceedings of the International Conference on Learning Representations*.

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

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

Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based sql query generation for cross-domain context-dependent questions. In *Proceedings of Empirical Methods in Natural Language Processing*.

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