arXiv:1807.03100v3 [cs.CL] 13 Sep 2018

Robust Text-to-SQL Generation with Execution-Guided Decoding

Chenglong Wang,[1]* Kedar Tatwawadi,[2]* Marc Brockschmidt,[3] Po-Sen Huang,[3]† Yi Mao,[3] Oleksandr Polozov,[3] Rishabh Singh[4]†
[1]University of Washington [2]Stanford University [3]Microsoft Research [4]Google Brain
clwang@cs.washington.edu kedart@stanford.edu
{mabrocks,pshuang,maoyi,polozov}@microsoft.com rising@google.com

Abstract

We consider the problem of neural semantic parsing, which translates natural language questions into *executable* SQL queries. We introduce a new mechanism, *execution guidance*, to leverage the semantics of SQL. It detects and excludes faulty programs during the decoding procedure by conditioning on the execution of partially generated program. The mechanism can be used with any autoregressive generative model, which we demonstrate on four state-of-the-art recurrent or template-based semantic parsing models. We demonstrate that execution guidance universally improves model performance on various text-to-SQL datasets with different scales and query complexity: WikiSQL, ATIS, and GeoQuery. As a result, we achieve new state-of-the-art execution accuracy of 83.8% on WikiSQL.

Introduction

Recent large-scale digitization of record keeping has resulted in vast databases that encapsulate much of an organization's knowledge. However, querying these databases usually requires users to understand specialized tools such as SQL, restricting access to this knowledge to select few. Thus, one aspiration of natural language processing is to translate natural language questions into formal queries that can be executed automatically and efficiently on a database, yielding natural interfaces to so-far inaccessible repositories of knowledge for end users. Developing effective semantic parsers to translate natural language questions into logical programs has been a long-standing goal (Poon 2013; Zettlemoyer and Collins 2005; Pasupat and Liang 2015; Li, Yang, and Jagadish 2005; Gulwani and Marron 2014). In this work, we focus on the semantic parsing task of translating natural language queries into executable SQL programs.

As in many other research areas, recently introduced deep learning approaches have been very successful in this task. A first generation of these approaches (e.g. by Iyyer, Yih, and Chang (2017)) has focused on adapting tools from (neural) machine translation, such as deep sequence-to-sequence (seq2seq) architectures with attention and copying mechanisms. While often effective, such approaches commonly fail at generating *syntactically valid* queries, and

*Equal contribution. Work done during two respective internships at Microsoft Research.
†Work done while at Microsoft Research.

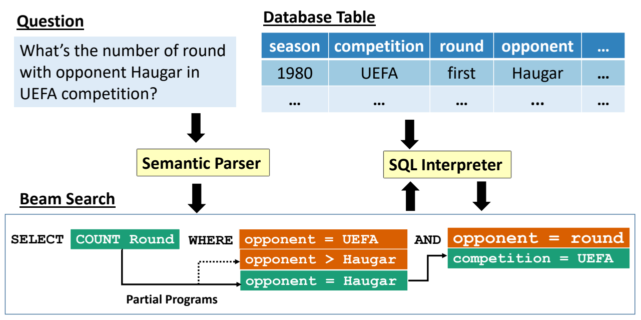


Figure 1: An execution-guided decoder evaluates partially generated queries at appropriate timesteps and then excludes those candidates that cannot be completed to a correct SQL query (red background). Here, "`opponent > Haugar`" would yield a runtime error, whereas "`opponent = UEFA`" would yield an empty result.

thus more recent work has shifted towards using grammar-based sequence-to-tree (seq2tree) models (Krishnamurthy, Dasigi, and Gardner 2017; Yin and Neubig 2017; Rabinovich, Stern, and Klein 2017; Xu, Liu, and Song 2017; Dong and Lapata 2018). In this work, we further extend this idea by showing how to condition such models to avoid whole classes of *semantic* errors, namely queries with runtime errors and queries that generate no results.

The key insight is that in languages such as SQL, a partially generated query can already be executed, and the results of that execution can be used to guide the generation procedure. We call this idea *execution guidance* and illustrate it in Fig. 1. Note that the concept can be extended beyond what we are demonstrating in this paper: whereas we only use execution of partial programs to filter out results that cannot be completed to a correct answer, a more advanced execution guidance mechanism could take partial results into account to improve decision-making, for example by considering which literals occur after filtering according to a partial query. In other words, execution guidance extends standard autoregressive decoders to additionally condition them on non-differentiable partial execution results at appropriate timesteps.

We show the effectiveness of execution guidance by extending a range of existing models with execution guidance and evaluating the resulting models on various text-to-SQL tasks with different scale and query complexity. Concretely, we first extend four state-of-the-art semantic parsing models with execution guidance and then demonstrate that execution guidance universally improves their performance on different text-to-SQL datasets. The considered models cover two widespread families of text-to-SQL semantic parsers: **(a)** autoregressive generative models such as Pointer-SQL (Wang, Brockschmidt, and Singh 2017; Wang et al. 2018) and Seq2Seq with attention (Iyer et al. 2017), as well as **(b)** template & slot-filling based models (a baseline by Finegan-Dollak et al. (2018) and Coarse2Fine by Dong and Lapata (2018)). We evaluate the unmodified baselines as well as our extensions on the WikiSQL (Zhong, Xiong, and Socher 2017), ATIS (Dahl et al. 1994), and GeoQuery (Zelle and Mooney 1996) datasets, showing that using the execution guidance paradigm during decoding leads to an improvement of $1\% - 6\%$ over the base models. As a result, our extension of Coarse2Fine with execution guidance becomes the state of the art model on the WikiSQL dataset with 83.8% execution accuracy.

Execution-Guided Decoding

As discussed above, the key insight in this work is that partially generated SQL queries can be executed to provide guidance for the remainder of the generation procedure. In this work, we only use this information to filter out partial results that cannot be completed to a correct query, for example because executing them yields a runtime error or because the generated query constraints already yield no results. We discuss these cases in detail now.

Execution Errors We consider the following two types of errors that could be identified by the execution engine:

- *Parsing errors*: A program p causes a parsing error if it is syntactically incorrect. This kind of error is more common for complex queries (as appearing in the GeoQuery and ATIS datasets). Autoregressive models are more prone to such errors than template-based and slot-filling-based models.
- *Runtime errors*: A program p throws a run-time error if it has a component whose operator type mismatches its operands types. Such an error could be caused by a mismatch between an aggregation function and its target column (e.g., sum over a column with string type) or a mismatch between a condition operator and its operands (e.g., applying $>$ to a column of float type and a constant of string type).

In these cases, the decoded program cannot be executed, and hence cannot yield a correct answer. If we can assume that every query *has* to yield a result, we consider an additional type of error:

- *Empty output*: When executed, a program p could return a empty result if the predicate generated by the decoder is overly restrictive (e.g., a predicate $c = v$ is generated but the constant v does not exist in the column c).

Algorithm 1 Execution-guided decoding as an extension of a standard recurrent autoregressive decoder.

```
1:  procedure EG-DECODING(encoded query $O$, encoded table columns $C$, beam size $k$)
2:      $\mathbf{h}_0 \leftarrow$ an initial hidden decoder state
3:      for all steps $1 \leq t \leq T$ of $k$-beam decoding do
4:          # Compute $k$ new decoder states in the beam:
5:          $\mathbf{h}_t^1, \ldots, \mathbf{h}_t^k \leftarrow \textsc{Decode}(O, C, \mathbf{h}_{t-1}^1, \ldots, \mathbf{h}_{t-1}^k)$
6:          $P_t^1, \ldots, P_t^k \leftarrow$ partial programs corresponding to the states $\mathbf{h}_t^1, \ldots, \mathbf{h}_t^k$
7:          if the current stage $t$ is executable then
8:              # Retain only the top $k$ executable programs:
9:              for $1 \leq i \leq k$ do
10:                 if $P_t^i$ has an error or empty output then
11:                     Remove $\mathbf{h}_t^i$ from the beam
12:     return the top-scored program $\operatorname{argmax}_{1 \leq i \leq k} \Pr(P_T^i)$
```

Note that in real-world situations it is common to expect that the query output is non-empty. For example, data scientists and database administrators often experiment with partial queries in order to compose a desired query from subtables, and use non-empty result as an indicator of the query's correctness (Iyyer, Yih, and Chang 2017).

Using Execution Guidance in Decoding To avoid generating queries yielding the errors discussed above, we integrate the query generation procedure with a SQL execution component. Extending a model with execution guidance thus requires to pick specific stages of the generative procedure at which to execute the partial result, and then use the result to refine the remaining generation procedure.

We show the pseudocode of an execution-guided extension of a standard autoregressive recurrent decoder in Alg. 1. It can be viewed as an extension of standard beam search applied to a model-specific decoder cell DECODE. Whenever possible (i.e., when the result at the current timestep t corresponds to an executable partial program), the procedure retains only the top k states in the beam that correspond to the partial programs without execution errors or empty outputs.

In non-autoregressive models based on feedforward networks, execution guidance can be used as a filtering step at the end of the decoding process, e.g., by dropping result programs that yield execution errors. The same can be applied to any autoregressive decoder at the end of beam decoding. However, in many application domains (including SQL generation), it is possible to apply execution checks to *partially* decoded programs, and not just at the end of beam decoding. For example, this allows to eliminate an incorrectly generated string-to-string inequality comparison "`... WHERE opponent > 'Haugar' ...`" from the beam immediately after the token `'Haugar'` is emitted (see Figure 1) As our experiments show, this significantly improves the effectiveness of execution guidance. The exact frequency and stages where execution guidance can be applied depends on the underlying decoder model.

Base Models

In this section, we describe the base models that we augmented with execution guidance and the details of our execution-guided decoder implementation for them. We only provide high-level model descriptions and refer to the respective source papers for details. In total, we extended four base models from prior work, which at the time of evaluation had achieved state-of-the-art performance on the WikiSQL, ATIS, or GeoQuery datasets. Two of them (Wang, Brockschmidt, and Singh 2017; Iyer et al. 2017) are *generative*, employing a recurrent sequence-to-sequence decoder. The other two (Dong and Lapata 2018; Finegan-Dollak et al. 2018) are *slot-filling*, employing a decoder that fills in the holes in a template (which itself can be generated with a sequence-to-sequence model). These models illustrate a wide variety of autoregressive decoders that can be augmented with execution guidance.

Seq2Seq with Attention (Iyer et al. 2017)

Iyer et al. (2017) apply a relatively standard sequence-to-sequence model to text-to-SQL tasks. The natural language input is encoded using a bidirectional RNN with LSTM (Hochreiter and Schmidhuber 1997) cells, and the target SQL query is directly predicted token by token. Additionally, an attention mechanism (Cho et al. 2014; Luong, Pham, and Manning 2015) is employed to improve the model performance. To handle the small number of data points in in some datasets, the encoding step uses pre-trained word embeddings from word2vec (Mikolov et al. 2013), which are concatenated to the embeddings that are learned for tokens from the training data.

Integrating Execution Guidance In classical sequence-to-sequence models, it is unclear a priori at which stages of the decoding we have a valid partial program. Thus, it is hard to implement partial program execution statically. We thus use the simplest form of execution-guidance: we first perform standard beam decoding with width k, and then choose the highest-ranked generated SQL program without execution errors at the end of the decoding.

Pointer-SQL (Wang, Brockschmidt, and Singh 2017)

The Pointer-SQL model introduced by Wang, Brockschmidt, and Singh (2017) extends and specializes the sequence-to-sequence architecture to the WikiSQL dataset. It takes a natural language question and a table schema of a single table t as inputs. To encode the inputs, a bidirectional RNN with LSTM cells is used to process the concatenation of the table header (column names) of the queried table and the question as input to learn a joint representation. The decoder is another RNN that can attend over and copy from the encoded input sequence. The key characteristic of this model is that the decoder uses three separate output modules corresponding to three decoding types. One module is used to generate SQL keywords, one module is used to copy a column name from the table header, and one module can copy literals from the natural language question.

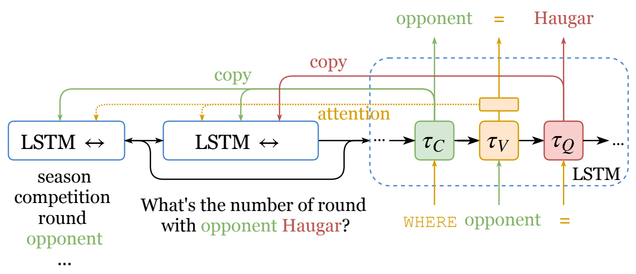


Figure 2: Overview of the Pointer-SQL model. The model encodes table columns as well as the user question with a BiLSTM and then decodes the hidden state with a typed LSTM, where the decoding action for each cell is statically determined. Source: Wang, Brockschmidt, and Singh (2017).

The grammar of SQL expressions in the WikiSQL dataset can be described by the regular expression "`Select` f c `From` t `Where` $(c\ op\ v)^*$". Here f refers to an aggregation function, c refers to a column name, t refers to the table name, op refers an comparator and v refers to a value. The Pointer-SQL model exploits this standardized shape of WikiSQL queries to select which output module to use at each decoding step, as shown in Figure 2.

Integrating Execution Guidance We extend the model to use execution guidance in two cases. First, after decoding the aggregation operator f and the aggregation column c, we run the execution engine over the partial program "`Select` f c `Where True`" to determine whether f and c are compatible. We replace decoding results failing this check by those pairs of f' and c' from the set of valid operator/column pairs with the next-highest joint probability according to the token distribution produced by the decoder; and then proceed to the decoding of predicates.

Second, after decoding a predicate $c_1\ op\ c_2$, we evaluate the partial program including the predicate to check whether the predicate triggers a type error or results in an empty output. Again, we replace decoding results failing this check by new predicates $c_1'\ op'\ c_2'$ with the next-highest joint probability from the set of error-free predicates.

In practice, instead of computing all correct choices, we parameterize the execution-guided decoder with a *beam width* k to restrict the number of alternative tokens considered at each decoding step and simply discard those results that trigger errors. As described earlier, this approach resembles a standard beam decoder where instead of generating the top-k results with the highest probability, we additionally use evaluation results to discard erroneous programs.

Template-Based Model (Finegan-Dollak et al. 2018)

The template-based approach to text-to-SQL generation introduced as baseline by Finegan-Dollak et al. (2018) also exploits the simple structure of target queries. First, the dataset is preprocessed to extract the most common *templates* – program sketches in which the operands in conditionals have been replaced by slots. The template-based model makes two kinds of predictions: **(a)** which template to use, and **(b)** which words in the question should be used to fill the slots in the chosen template. For this, a bidirectional RNN is run over the

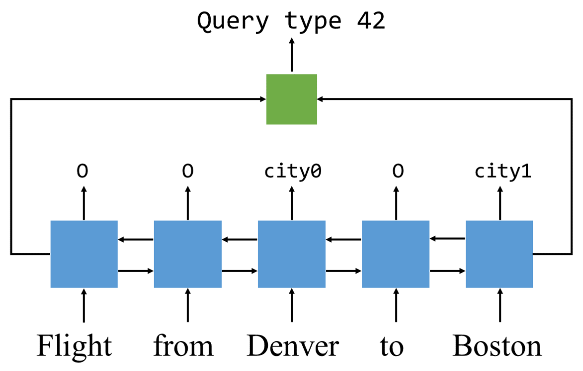


Figure 3: Overview of the template-based baseline model. The encoder consists of an Bi-LSTM, whose outputs are then used by feed-forward networks to determine the template and fill in the slots. Source: Finegan-Dollak et al. (2018).

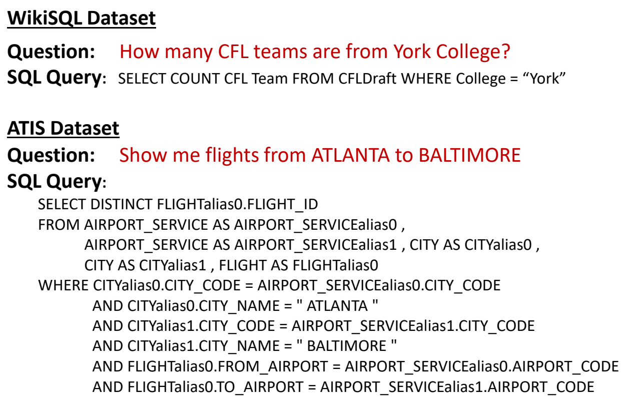
natural language question, outputting a “used in slot” or “not used in query” signal for each token. A small fully-connected network is then used to predict the chosen template from the final states of the RNN. The output query is constructed by filling the slots from the template with the predicted tokens from the input question; but as no agreement between the two predictions is enforced, this can easily fail. Figure 3 illustrates the structure of this template-based model. As the model is restricted to templates seen only during training, it cannot generalize to query structures that do not appear in the training set.

Integrating Execution Guidance For a given beam width of k, we independently pick the top-k choices for the template and for the slots using beam decoding. The final top-k template-slot candidate predictions are decided based on the joint probability of the template and the slots. We then pick the SQL program with the highest joint probability which does not lead to an execution error. Partial program execution is not possible in this model, as the model does not employ an autoregressive recurrent decoder.

Coarse2Fine (Dong and Lapata 2018)

The Coarse2Fine model can be viewed as a mix of template-based models and sequence-to-sequence models. It is a two-stage model for general text-to-code translation, where the first stage generates a coarse “sketch” (a template) of the target program and the second stage fills its missing “slots”.

The model generates programs in the following three steps. First, the input question (and in the case of WikiSQL, the table schema) is encoded using a bidirectional RNN using LSTM cells. Then, the *sketch generator* uses a classifier to choose a query sketch of the form “`Where` (op)*” from one of the predefined sketches. Intuitively, the sketch determines the number of conditions in the `Where`-clause as well as the comparison operators. Finally, the *fine meaning decoder* uses the inputs as well as the generated sketch to produce the full query by filling in slots. Similarly to Pointer-SQL, the fine-decoding model uses a copying mechanism to generate column names and constants.



WikiSQL Dataset

Question: How many CFL teams are from York College?

SQL Query: SELECT COUNT CFL Team FROM CFLDraft WHERE College = “York”

ATIS Dataset

Question: Show me flights from ATLANTA to BALTIMORE

SQL Query:

```
SELECT DISTINCT FLIGHTalias0.FLIGHT_ID
FROM AIRPORT_SERVICE AS AIRPORT_SERVICEalias0 ,
        AIRPORT_SERVICE AS AIRPORT_SERVICEalias1 , CITY AS CITYalias0 ,
        CITY AS CITYalias1 , FLIGHT AS FLIGHTalias0
WHERE CITYalias0.CITY_CODE = AIRPORT_SERVICEalias0.CITY_CODE
         AND CITYalias0.CITY_NAME = " ATLANTA "
         AND CITYalias1.CITY_CODE = AIRPORT_SERVICEalias1.CITY_CODE
         AND CITYalias1.CITY_NAME = " BALTIMORE "
         AND FLIGHTalias0.FROM_AIRPORT = AIRPORT_SERVICEalias0.AIRPORT_CODE
         AND FLIGHTalias0.TO_AIRPORT = AIRPORT_SERVICEalias1.AIRPORT_CODE
```

Figure 4: Example of WikiSQL and ATIS queries

Integrating Execution Guidance We cannot apply execution-guidance to the sketch generator of the Coarse2Fine model, as the generated sketch is not an executable partial program. Here, we simply pick the most probable sketch, and proceed to the next stage of decoding.

Similarly to the execution-guidance implementation for the Pointer-SQL model, after decoding the aggregation operator f and the aggregation column c we run the execution engine over the partial program “`Select` f c `From` t `Where True`”, and pick a *compatible* (f, c) pair with the highest joint probability.

In the slot-filling stage, when `Where` c_1 op c_2 templates are completed using the fine meaning decoder, we apply execution guidance in a similar way as in the Pointer-SQL model. As the op operations were previously generated as part of the sketch, we retain only the k highest-ranked (c_1, c_2) combinations that do not result in an execution error. If no valid choices are found, we backtrack and emit a different sketch from the “coarse” model.

Experimental Results

We evaluate the effect of execution guidance by comparing the baseline models with our new variants that integrate execution guidance. Because this guidance only requires changes to the generation procedure at test time, we can use the same pre-trained models in all instances. As discussed above, our four different models cover two different families (generative and slot-filling) and integrate execution guidance to different degrees.

For each dataset, we pick a model from each family that has achieved state-of-the-art performance at the time of evaluation. Thus, specifically, we evaluate the Pointer-SQL model (Wang, Brockschmidt, and Singh 2017) and the Coarse2Fine (Dong and Lapata 2018) model on the WikiSQL dataset. The template-based baseline (Finegan-Dollak et al. 2018) and the Seq2Seq with Attention (Iyer et al. 2017) models are evaluated on the ATIS and GeoQuery datasets.

WikiSQL Experiments

WikiSQL (Zhong, Xiong, and Socher 2017) is a recently introduced natural language to SQL dataset, consisting of

Model	Dev		Test	
	Acc_{syn}	Acc_{ex}	Acc_{syn}	Acc_{ex}
Pointer-SQL (2017)	61.8	72.5	62.3	71.9
Pointer-SQL + EG (3)	66.6	77.3	66.7	76.9
Pointer-SQL + EG (5)	**67.5**	**78.4**	**67.9**	**78.3**
Coarse2Fine (2018)	72.9	79.2	71.7	78.4
Coarse2Fine + EG (3)	75.6	83.4	74.8	83.0
Coarse2Fine + EG (5)	**76.0**	**84.0**	**75.4**	**83.8**

Table 1: Test and Dev accuracy (%) of the models on WikiSQL data, where Acc_{syn} refers to syntactical accuracy and Acc_{ex} refers to execution accuracy. "+ EG (k)" indicates that model outputs are generated using the execution-guided strategy with beam size k.

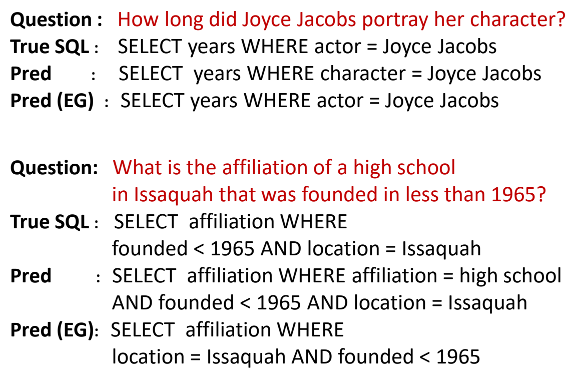
80,654 pairs of questions and SQL queries distributed across 24,241 tables from Wikipedia. The task is to generate the correct SQL query for a given natural language question and a table schema (i.e., table column names), without using the content values of tables. The SQL structure in the WikiSQL dataset is simple and always follows the structure `SELECT agg sel WHERE (col op cond)*`. Also, the natural language question is often grammatically wrong (see Figure 1 for an example). Nonetheless, the WikiSQL dataset poses a good challenge for text-to-SQL systems and has seen significant interest since its release (Dong and Lapata 2018; Wang et al. 2018; Huang et al. 2018; Yu et al. 2018). In our experiments, we use the default train/development/test split, leading to 56,324 training pairs, 8,421 development pairs, and 15,878 test pairs. Each table is present in only one split to test generalization to unseen tables.

Table 1 shows the results for Pointer-SQL and Coarse2Fine. We report both the syntactical accuracy Acc_{syn} corresponding to the ratio of predictions that are exactly the ground truth SQL query, as well as the execution accuracy Acc_{ex} corresponding to the ratio of predictions that return the same result as the ground truth when executed. Note that the execution accuracy is higher than syntactical accuracy as syntactically different programs can generate the same results (e.g., programs differing only in predicate order). In execution-guided decoding, we report two model variants, one using a beam size of 3 and the other a beam size of 5.

The comparison results show that execution-guided decoding significantly improves both syntactical accuracies as well as execution accuracies of the models. Using a beam size of 5 leads to an improvement of 6.4% (71.9% to 78.3%) for the Pointer-SQL model, and an improvement of 5.4% (78.4% to 83.8%) for the Coarse2Fine model on the Test dataset (see Figure 5 for some examples). Similar improvements are observed on the Dev dataset. To the best of our knowledge, this improvement on the Coarse2Fine model makes it the new state of the art in terms of execution accuracy on WikiSQL.

ATIS and GeoQuery Experiments

We use the ATIS and GeoQuery datasets from the standardized collection of datasets (Finegan-Dollak et al. 2018, ver. 1) on the default train/test/dev split. Compared to the WikiSQL


Question : How long did Joyce Jacobs portray her character?
True SQL : SELECT years WHERE actor = Joyce Jacobs
Pred : SELECT years WHERE character = Joyce Jacobs
Pred (EG) : SELECT years WHERE actor = Joyce Jacobs

Question: What is the affiliation of a high school in Issaquah that was founded in less than 1965?
True SQL : SELECT affiliation WHERE founded < 1965 AND location = Issaquah
Pred : SELECT affiliation WHERE affiliation = high school AND founded < 1965 AND location = Issaquah
Pred (EG): SELECT affiliation WHERE location = Issaquah AND founded < 1965

Figure 5: Some examples where execution guidance (EG) for Coarse2Fine leads to correct prediction. In the first example, the table column is corrected by execution guidance due to an empty output. In the second example, the execution guidance corrects the sketch as all possible slot-filling options for the three-condition sketch overconstrain the program and also yield an empty output. The experiments were performed with beam size of 5.

data, these datasets consider substantially more complex SQL queries, for example requiring to access multiple tables in one SQL query. The ATIS dataset on average references 6 tables per query, while the GeoQuery dataset on average references 3 tables per query. There are also other complexities in the SQL programs, such as nesting, inner joins, and group-by operations. See Figure 4 for an example.

Table 2 shows the results for the template-based model (Finegan-Dollak et al. 2018) and the sequence-to-sequence model (Iyer et al. 2017) for the ATIS and GeoQuery datasets. We compare the performance of the base models with execution guided decoding with beam size 5 and 10. We report execution accuracy for evaluation on both the dev and test splits of the data.

The comparison result shows that execution-guided decoding leads to an improvement ranging from 0.4% to 5% on the test data. The template-based model improves most by 5% on the ATIS dataset, and by 0.4% on the GeoQuery dataset. The significantly smaller improvement can be attributed to the slot-filling nature of the model (which is less amenable to partial execution guidance) along with the simpler structure of the GeoQuery dataset. Conversely, the sequence-to-sequence model sees an improvement of 2.5% on the GeoQuery dataset and only a 0.9% improvement on the ATIS dataset.

Discussion & Analysis

Our experiments show that execution guidance is a simple but effective tool to improve a wide range of existing text-to-SQL models. However, we note that this strategy simply improves the number of semantically meaningful programs and not only the number of semantically correct programs. For this, consider Tables 3 and 4, in which we show the number of execution errors remaining after adding execution guidance to the models. For the Coarse2Fine model on the

Model	ATIS		GeoQuery	
	Dev Acc_{ex}	Test Acc_{ex}	Dev Acc_{ex}	Test Acc_{ex}
Template-based (2018)	35.1	32.6	50.5	55.2
Template-based + EG (5)	36.7	37.1	**52.7**	55.2
Template-based + EG (10)	**37.6**	**37.6**	**52.7**	**55.6**
Seq2Seq (2017)	78.6	77.0	76.0	72.5
Seq2Seq + EG (5)	**78.8**	77.3	**78.0**	**75.0**
Seq2Seq + EG (10)	**78.8**	**77.9**	**78.0**	**75.0**

Table 2: Test and Dev accuracy (%) of the models on ATIS and GeoQuery data, where Acc_{ex} refers to execution accuracy. "+ EG (k)" indicates that model outputs are generated using the execution guiding strategy with beam size k.[1]

Model	Acc_{ex}	Execution Errors
Coarse2Fine (2018)	78.4	10.70
Coarse2Fine + EG (3)	83.0	0.03
Coarse2Fine + EG (5)	83.8	0.01

Table 3: Accuracy (%) and the overall amount of execution errors (%) on the WikiSQL test dataset.

Model	Acc_{ex}	Execution Errors
Seq2Seq (2017)	77.0	10.5
Seq2Seq + EG (5)	77.3	6.9
Seq2Seq + EG (10)	77.9	6.3

Table 4: Accuracy (%) and the overall amount of execution errors (%) on the ATIS test dataset.

Model	Acc_{ex} (%)
Coarse2Fine (2018)	78.4
Coarse2Fine + EG (5)	**83.8**
No Aggregation execution	83.7
No Condition execution	80.2
No Sketch backtracking	82.6

Table 5: Ablation study on different execution-guided decoding steps on the WikiSQL dataset.

WikiSQL data, we see a drop of 10% in execution errors, but only an improvement of execution accuracy of 5.4%. Thus, about half of the incorrect programs were replaced by correct predictions. On the other hand, the Seq2Seq model on the ATIS data sees a drop of execution errors of only 4.2% (the large number of remaining errors is due to the more complex nature of ATIS SQL queries). More significantly, this only yields an improvement in accuracy of 0.9%. Note that our integration of execution guidance for this model is only post-hoc filtering, and thus we speculate that more fine-grained execution checks on partial programs is more effective.

Ablations To better understand the source of improvement from execution guidance, we performed additional ablation experiments on the Coarse2Fine model. Recall that the execution guidance on the Coarse2Fine model applies at three different intermediate steps of the decoding process. When the model generates a WikiSQL query of form "`Select` f c `From` t `Where` (c op v)*", the execution-guided decoder **(a)** checks the choice of the aggregation f c, **(b)** checks the choice of each generated condition c op v, and **(c)** backtracks to a different sketch from the "coarse" stage of the Coarse2Fine model if no generated candidate programs execute correctly. We perform ablation experiments in which we turn off execution guidance at each of these steps respectively. We use beam size of 5 for all the ablation experiments.

The results are shown in Table 5 and show that execution guidance has little effect on the choice of aggregation functions, but contributes significantly in the generation of conditions. This experiment further emphasizes the importance of fine-grained guidance that eliminates incorrect partial candidates at intermediate decoding steps.

Related Work

Semantic Parsing Semantic parsing has been studied extensively by the natural language processing community. It maps natural language to a logical form representing its meaning, which is then used for question answering (Wong and Mooney 2007; Zettlemoyer and Collins 2005; 2007; Berant et al. 2013), robot navigation (Chen and Mooney 2011; Tellex et al. 2011), and many other tasks. Liang (2016) surveys different statistical semantic parsers and categorizes them under a unified framework.

Recently there has been an increasing interest in applying deep learning models to semantic parsing, due to the huge success of such models on machine translation and other problems. A significant amount of work is dedicated to constraining model outputs to guarantee valid parsing results. Dong and Lapata (2016) propose the Seq2Tree model, which always generates syntactically valid trees. The same authors

[1]The execution accuracy results of the template-based model are lower than originally reported by Finegan-Dollak et al. (2018), because at the time of writing, a bug was discovered in the evaluation of the original model. Finegan-Dollak et al. later made changes to their template-based model. We used their published pre-trained models for our experiments.

later propose a two-step decoding approach which first generates a rough sketch, and later uses it to constrain the final output during the second decoding pass (Dong and Lapata 2018). In contrast to these token-based decoding approaches, the work of Yin and Neubig (2017) and Krishnamurthy, Dasigi, and Gardner (2017) employ grammar-based decoding which utilizes grammar production rules as a decoding constraint.

The use of SQL as the meaning representation for semantic parsing was recently re-popularized by the introduction of the WikiSQL dataset (Zhong, Xiong, and Socher 2017). A large number of subsequent neural semantic parsing works included an evaluation on WikiSQL (Xu, Liu, and Song 2017; Huang et al. 2018; Yu et al. 2018; Wang et al. 2018; Dong and Lapata 2018). The large number of annotated pairs of natural language queries together with their corresponding SQL representations makes it an attractive dataset for training neural network models. However, WikiSQL is often criticized for its weak coverage of SQL syntax. Finegan-Dollak et al. (2018) later standardized a range of semantic parsing datasets with SQL as the logical representation, which supports more realistic usage of language including table joins and nested queries. We use their standardization of the ATIS and GeoQuery datasets in our experiments.

Sequence-Level Objectives Several recently introduced techniques use reinforcement learning to incorporate sequence-level objectives into training, such as BLEU in coherent text generation (Bosselut et al. 2018), semantic coherence in dialogue generation (Li et al. 2016), and SPIDEr in image captioning (Liu et al. 2017). Similarly, Zhong, Xiong, and Socher (2017) use reinforcement learning to learn a policy with the objective of maximizing the expected correctness of the execution of generated programs. Wiseman and Rush (2016) propose incorporating sequence-level cost functions (such as BLEU) into beam search for sequence-to-sequence training. These efforts are focused on integrating additional (discrete) objectives into the training objective, whereas our work only requires changes during inference time and can thus easily be applied to a wide range of different models, as shown in our experiments.

Program Synthesis The related research area of program synthesis aims to generate programs given some specification of user intent such as input-output examples or natural language descriptions (Gulwani, Polozov, and Singh 2017). The latest work in this field combines neural and symbolic techniques in order to generate programs that are both likely to satisfy the specification and semantically correct. This line of work includes DeepCoder (Balog et al. 2017), Neurosymbolic synthesis (Parisotto et al. 2017), Neo (Feng et al. 2018), and NGDS (Kalyan et al. 2018). They use probabilistic models to guide symbolic program search to maximize the likelihood of generating a program suitable for the specification. Our execution guidance idea is similar to neurosymbolic program synthesis, but in contrast, it guides a neural program generation model using a symbolic component (namely, partial program execution) to generate semantically meaningful programs.

Conclusion

For the task of SQL generation from natural language, we presented the idea of guiding the generation procedure by partial execution results. This approach allows conditioning an arbitrary autoregressive decoder on non-differentiable partial execution results at inference time, leading to elimination of semantically invalid programs from the candidates. We showed the widespread practical utility of execution guidance by applying it on four different state-of-the-art models which we then evaluated on three different datasets. The improvement offered by execution-guided decoding is dependent on the nature of the extended model and the degree of integration with the decoding procedure. Extending the Coarse2Fine model with an execution-guided decoder improves its accuracy by 5.4% (from 78.4% to 83.8%) on the WikiSQL test dataset, making it the new state of the art on this task.

The idea of execution guidance can potentially be applied to other tasks, such as natural language to logical form, which we plan to explore in future work. Furthermore, we note that in this work, we only used execution guidance to filter out incorrect predictions during generation at inference time. We believe that integrating execution guidance into the training phase, for example by learning to make decisions conditional upon the results of executing the partial program generated so far, will further improve model performance.

References

Balog, M.; Gaunt, A. L.; Brockschmidt, M.; Nowozin, S.; and Tarlow, D. 2017. DeepCoder: Learning to write programs. In *International Conference on Learning Representations*.

Berant, J.; Chou, A.; Frostig, R.; and Liang, P. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing*.

Bosselut, A.; Celikyilmaz, A.; He, X.; Gao, J.; Huang, P.; and Choi, Y. 2018. Discourse-aware neural rewards for coherent text generation. In *North American Chapter of the Association for Computational Linguistics*.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Dahl, D. A.; Bates, M.; Brown, M.; Fisher, W.; Hunicke-Smith, K.; Pallett, D.; Pao, C.; Rudnicky, A.; and Shriber, E. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. *Proceedings of the workshop on Human Language Technology* 43–48.

Dong, L., and Lapata, M. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Dong, L., and Lapata, M. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Feng, Y.; Martins, R.; Bastani, O.; and Dillig, I. 2018. Program synthesis using conflict-driven learning. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 420–435. ACM.

Finegan-Dollak, C.; Kummerfeld, J. K.; Zhang, L.; Ramanathan, K.; Sadasivam, S.; Zhang, R.; and Radev, D. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Gulwani, S., and Marron, M. 2014. NLyze: interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*.

Gulwani, S.; Polozov, O.; and Singh, R. 2017. Program synthesis. *Foundations and Trends® in Programming Languages* 4(1-2):1–119.

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

Huang, P.-S.; Wang, C.; Singh, R.; Yih, W.-t.; and He, X. 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*.

Iyer, S.; Konstas, I.; Cheung, A.; Krishnamurthy, J.; and Zettlemoyer, L. 2017. Learning a neural semantic parser from user feedback. In *ACL*, 963–973.

Iyyer, M.; Yih, S. W.-T.; and Chang, M.-W. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Kalyan, A.; Mohta, A.; Polozov, O.; Batra, D.; Jain, P.; and Gulwani, S. 2018. Neural-guided deductive search for real-time program synthesis from examples. In *International Conference on Learning Representations*.

Krishnamurthy, J.; Dasigi, P.; and Gardner, M. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 1516–1526.

Li, J.; Monroe, W.; Ritter, A.; Galley, M.; Gao, J.; and Jurafsky, D. 2016. Deep reinforcement learning for dialogue generation. In *EMNLP*.

Li, Y.; Yang, H.; and Jagadish, H. V. 2005. NaLIX: An interactive natural language interface for querying XML. In *SIGMOD*, 900–902.

Liang, P. 2016. Learning executable semantic parsers for natural language understanding. *Communications of the ACM* 59.

Liu, S.; Zhu, Z.; Ye, N.; Guadarrama, S.; and Murphy, K. 2017. Improved image captioning via policy gradient optimization of SPIDEr. In *ICCV*.

Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 3111–3119.

Parisotto, E.; Mohamed, A.; Singh, R.; Li, L.; Zhou, D.; and Kohli, P. 2017. Neuro-symbolic program synthesis. In *International Conference on Learning Representations*.

Pasupat, P., and Liang, P. 2015. Compositional semantic parsing on semi-structured tables. In *ACL*, 1470–1480.

Poon, H. 2013. Grounded unsupervised semantic parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistic*, 933–943.

Rabinovich, M.; Stern, M.; and Klein, D. 2017. Abstract syntax networks for code generation and semantic parsing. *CoRR* abs/1704.07535.

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M. R.; Banerjee, A. G.; Teller, S.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the National Conference on Artificial Intelligence*, 1507–1514.

Wang, C.; Huang, P.-S.; Polozov, O.; Brockschmidt, M.; and Singh, R. 2018. Execution-guided neural program decoding. *arXiv preprint arXiv:1807.03100*.

Wang, C.; Brockschmidt, M.; and Singh, R. 2017. Pointing out SQL queries from text. Technical Report MSR-TR-2017-45, Microsoft Research.

Wiseman, S., and Rush, A. M. 2016. Sequence-to-sequence learning as beam-search optimization. In *EMNLP*.

Wong, Y. W., and Mooney, R. J. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*.

Xu, X.; Liu, C.; and Song, D. 2017. SQLNet: Generating structured queries from natural language without reinforcement learning. *CoRR* abs/1711.04436.

Yin, P., and Neubig, G. 2017. A syntactic neural model for general-purpose code generation. *CoRR* abs/1704.01696.

Yu, T.; Li, Z.; Zhang, Z.; Zhang, R.; and Radev, D. 2018. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In *NAACL-HLT*, 588–594.

Zelle, J. M., and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, 1050–1055.

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

Zettlemoyer, L. S., and Collins, M. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 678–687.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *ArXiv e-prints*.