INCSQL: TRAINING INCREMENTAL TEXT-TO-SQL PARSERS WITH NON-DETERMINISTIC ORACLES

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a sequence-to-action parsing approach for the natural language to SQL task that incrementally fills the slots of a SQL query with feasible actions from a pre-defined inventory. To account for the fact that typically there are multiple correct SQL queries with the same or very similar semantics, we draw inspiration from syntactic parsing techniques and propose to train our sequence-to-action models with non-deterministic oracles. We evaluate our models on the WikiSQL dataset and achieve an execution accuracy of 83.7% on the test set, a 2.1% absolute improvement over the models trained with traditional static oracles assuming a single correct target SQL query. When further combined with the execution-guided decoding strategy, our model sets a new state-of-the-art performance at an execution accuracy of 87.1%.

1 INTRODUCTION

Many mission-critical applications in health care, financial markets, and business process management store their information in relational databases (Hillestad et al., 2005; Ngai et al., 2009; Levine et al., 2001). Users access that information using a query language such as SQL. Although expressive and powerful, SQL is difficult to master for non-technical users. Even for an expert, writing SQL queries can be challenging as it requires knowing the exact schema of the database and the roles of various entities in the query. Hence, a long-standing goal has been to allow users to interact with the database through natural language (Androutsopoulos et al., 1995; Popescu et al., 2003).

The key to achieving this goal is understanding the semantics of the natural language statements and mapping them to the intended SQL. This problem, also known as NL2SQL, was previously understudied largely due to the availability of annotation. Without paired natural language statement and SQL query, a weak supervision approach may be adopted which reduces supervision from annotated SQL queries to answers (Liang et al., 2011). This is a more difficult learning problem. Therefore only with recent release of a number of large-scale annotated NL2SQL datasets (Zhong et al., 2017; Finegan-Dollak et al., 2018), we start to see a surge of interest in solving this problem.

Existing NL2SQL approaches largely fall into two categories: sequence-to-sequence style neural "machine translation" systems (Zhong et al., 2017; Dong & Lapata, 2018) and sets of modularized models with each predicting a specific part of the SQL queries (Xu et al., 2017; Yu et al., 2018). The former class suffer from the requirement of labeling a single ground truth query while multiple semantically equivalent queries exist for each intent. For example, as noticed by Zhong et al. (2017), the ordering of filtering conditions in a query does not affect execution but affects generation. To account for this, techniques such as reinforcement learning have been used on top of those sequence-to-sequence models. The second class of models employ a sequence-to-set approach: they first predict table columns present in the query and then independently predict the rest for each column. This avoids the ordering issue, but makes it harder to leverage inter-dependencies among conditions.

In this work, we develop a sequence-to-action parsing approach (Section 3) for the NL2SQL problem. It incrementally fills the slots of a SQL query with actions from an inventory designed for this task. Taking inspiration from training oracles in incremental syntactic parsing (Goldberg & Nivre, 2013), we further propose to use non-deterministic oracles (Section 4) for training the incremental parsers. These oracles permit multiple correct action continuations from a partial parse, thus are able to account for the logical form variations. Our model combines the advantage of a sequence-to-sequence model that captures inter-dependencies within sequence of predictions and a

Input x:　　　　　　What is the height of Willis Tower in Chicago?

Rank	Name	Location	Height (ft)	Floor	Year
1	One World Trade Center	New York City	1,776	104	2014
2	Willis Tower	Chicago	1,451	108	1974
...	...	...	...	...	...

Output y:

```
SELECT `Height (ft)`
WHERE Name="Willis Tower"
  AND Location="Chicago"
```

Execution Result:　　1,451

Figure 1: Our running example. The input is a natural language question and a table schema, and the output is an executable SQL query. Table contents are shown here, but unknown to our models.

modularized model that avoids any standarized linearization of the logical forms. We evaluate our models on the WikiSQL dataset and observe a performance improvement of 2.1% when comparing non-deterministic oracles with traditional static oracles. We further combine our approach and the execution-guided decoding strategy (Wang et al., 2018) and achieve a new state-of-the-art performance with 87.1% test execution accuracy. Experiments on a filtered ATIS dataset in addition confirm that our models can be applied to other NL2SQL datasets.

2　Task Definition

Given an input natural language question, our goal is to generate its corresponding SQL query. In the following and throughout the paper, we use the WikiSQL dataset (Zhong et al., 2017) as our motivating example. However, it should be noted that our approach is generally applicable to other NL2SQL data, with proper choice of an action inventory and redesign of parser states.

WikiSQL dataset consists of 80,654 pairs of questions and SQL queries distributed across 24,241 tables from Wikipedia. Along with the natural language question, the input also contains a single table schema (i.e., table column names). Each table is present in only one split (train, dev, or test), which demands the models to generalize to unseen tables. Figure 1 shows an example.

The SQL structure of the WikiSQL dataset queries is restricted and always follows the template `SELECT agg selcol WHERE col op val (AND col op val)*`. Here, `selcol` is a single table column and `agg` is an aggregator (e.g., `COUNT`, `SUM`, empty). The `WHERE` segment is a sequence of conjunctive filtering conditions. Each `op` is a filtering operator (e.g., $=$) and the filtering value `val` is mentioned in the question. Although the dataset comes with a "standard" linear ordering of the conditions, the order is actually irrelevant given the semantics of `AND`.

Throughout the paper we denote the input to the parser as x. It consists of a natural language question w with tokens w_i and a single table schema c with column names c_j. A column name c_j can have one or more tokens. The parser needs to generate an executable SQL query y as its output.

3　Our Model: An Incremental Parser

Given an input x, the generation of a structured output y is broken down into a sequence of parsing decisions. The parser starts from an initial state and incrementally takes actions according to a learned policy. Each action advances the parser from one state to another, until it reaches one of the terminal states, where we may extract a complete logical form y. We take a probabilistic approach to model the policy. It predicts a probability distribution over the valid set of subsequent actions given the input x and the running decoding history. The goal of training such an incremental semantic parser is then to optimize this parameterized policy.

Formally, we let $P_\theta(y|x) = P_\theta(\boldsymbol{a}|x)$, where θ is model parameters. Execution of the action sequence $\boldsymbol{a} = \{a_1, a_2, \ldots, a_k\}$ leads the parser from the initial state to a terminal state that contains the parsing result y. Here we assume that each y has only one corresponding action sequence $\boldsymbol{a}$, an assumption that we will revisit in Section 4. The probability of action sequence is further factored as the product of incremental decision probabilities: $P_\theta(\boldsymbol{a}|x) = \prod_{i=1}^{k} P_\theta(a_i|x, a_{<i})$, where $|\boldsymbol{a}| = k$. During inference, instead of attempting to enumerate over the entire output space and find the highest scoring $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} P_\theta(\boldsymbol{a}|x)$, our decoder takes a greedy approach: at each intermediate step, it picks the highest scoring action according to the policy: $a_i^* = \arg\max_{a_i} P_\theta(a_i|x, a_{<i}^*)$.

Action	Resulting state after taking the action at state p	Parameter representation
AGG(agg)	$p[\text{AGG}\mapsto\text{agg}]$	–
SELCOL(c_i)	$p[\text{SELCOL}\mapsto c_i]$	$\boldsymbol{r}_i^C$
CONDCOL(c_i)	$p[\text{COND}\mapsto p.\text{COND}\|\| \begin{bmatrix} \text{COL} & c_i \\ \text{OP} & \epsilon \\ \text{VAL} & \epsilon \end{bmatrix}]$	$\boldsymbol{r}_i^C$
CONDOP(op)	$p[\text{COND}_{-1}\mapsto p.\text{COND}_{-1}[\text{OP}\mapsto\text{op}]]$	–
CONDVAL($w_{i:j}$)	$p[\text{COND}_{-1}\mapsto p.\text{COND}_{-1}[\text{VAL}\mapsto w_{i:j}]]$	$\boldsymbol{r}_i^W$ and $\boldsymbol{r}_j^W$
END	p (as a terminal state)	–

Table 1: The inventory of actions for our incremental text-to-SQL semantic parser. The use of parameter representations in the decoder is detailed in Section 3.2. $p[\text{AGG}\mapsto\text{agg}]$ denotes a new parser state identical to p, except that the feature value for AGG is agg in the new state. Finally, $\|\|$ denotes list concatenation, and COND_{-1} refers to the last element in the list.

In the following subsections, we define the parser states and the inventory of actions, followed by a description of our encoder-decoder neural-network model architecture.

3.1 SEQUENCE TO ACTIONS

We first look at a structured representation of a full parse corresponding to the example in Figure 1:

$$\begin{bmatrix} \text{AGG} & \text{NONE} \\ \text{SELCOL} & \text{Height (ft)} \\ \\ \text{COND} & \left\langle \begin{bmatrix} \text{COL} & \text{Name} \\ \text{OP} & = \\ \text{VAL} & \text{``Willis Tower''} \end{bmatrix}, \begin{bmatrix} \text{COL} & \text{Location} \\ \text{OP} & = \\ \text{VAL} & \text{``Chicago''} \end{bmatrix} \right\rangle \end{bmatrix}.$$

An intermediate parser state is thus defined as a partial representation, with some feature values not filled in yet, denoted as ϵ. The initial parser state p_0 has only empty list and values $\begin{bmatrix} \text{AGG} & \epsilon \\ \text{SELCOL} & \epsilon \\ \text{COND} & \langle\rangle \end{bmatrix}$.

Next, we define our inventory of actions. Each action has the effect of advancing the parser state from p to p'. We let $p = \begin{bmatrix} \text{AGG} & \text{agg} \\ \text{SELCOL} & \text{selcol} \\ \text{COND} & \text{cond} \end{bmatrix}$, and describe p' for each action in Table 1. The action CONDVAL selects a span of text $w_{i:j}$ from the input question w. In practice, this leads to a large number of actions, quadratic in the length of the input question, so we break down CONDVAL into two consecutive actions, one selecting the starting position w_i and the other selecting the ending position w_j for the span. At the end of the action sequence, we append a special action END that terminates the parsing process and brings the parser into a terminal state. As an example, the query in Figure 1 translates to an action sequence of { AGG(NONE), SELCOL(c_3), CONDCOL(c_1), CONDOP(=), CONDVAL($w_{5:6}$), CONDCOL(c_2), CONDOP(=), CONDVAL($w_{8:8}$)}.

The above definitions assume all the valid sequences to have the form of AGG SELCOL (CONDCOL CONDOP CONDVAL)* END. This guarantees that we can extract a complete logical form from each terminal state. For other data with different SQL structure, a redesign of action inventory and parser states is required.

3.2 DECODER

We first assume that we have some context-sensitive representations $\boldsymbol{r}_i^W$, $\boldsymbol{r}_j^C$ for each word w_i and each column header c_j, respectively, and detail our design of the decoder here. The encoder for obtaining the representations $\boldsymbol{r}_i^W$, $\boldsymbol{r}_j^C$ will be discussed in Section 3.3.
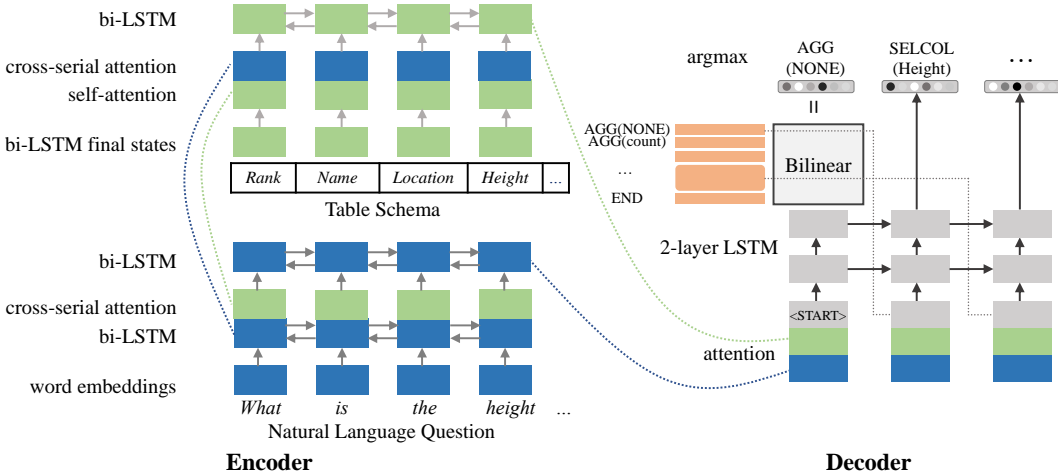
Figure 2: Our main model architecture depicted using our running example. The decoder (right) and the encoder (left) are described in Sections 3.2 and 3.3, respectively.

The main component of our decoder is to model a probability distribution $P_\theta(a|x, a_{<i})$ over potential parser actions a conditioned on input x and past actions $a_{<i}$. It has two main challenges: (1) there is no fixed set of valid parser actions: it depends on the input and the current parser state; (2) the parser decision is context-dependent: it relies on the decoding history and the information embedded in the input question and column headers.

We adopt an LSTM-based decoder framework and address the first challenge through individual scoring of actions. The model scores each candidate action a as s_a and uses a softmax function to normalize the scores into a probability distribution. At time step i, we denote the current decoder hidden state as $\boldsymbol{h}_i^{\text{DEC}}$ and model the score of a in the form of a bilinear function: $s_a = (\boldsymbol{h}_i^{\text{DEC}})^T U^A \boldsymbol{r}_a^A$, where $\boldsymbol{r}_a^A$ is a vector representation of the action a and is modeled as the concatenation of the action embedding and the parameter representation. The form of the latter is given in Table 1.

The dependencies between the parser decisions and the input question and column headers are captured through a dot-product attention mechanism (Luong et al., 2015). The input to the first layer of our decoder LSTM at time step $i + 1$ is a concatenation of the output action representation $\boldsymbol{r}_{a_i}^A$ from previous time step i, a question attention vector $\boldsymbol{e}_i^W$, and a column header attention vector $\boldsymbol{e}_i^C$. Vector $\boldsymbol{e}_i^C = \sum_j \alpha_{i,j} \boldsymbol{r}_j^C$, where $\alpha_{i,j} \propto \boldsymbol{h}_i^{\text{DEC}} \cdot \boldsymbol{r}_j^C$. Vector $\boldsymbol{e}_i^W$ is defined similarly.

3.3 ENCODER

Now we return to the context-sensitive representations $\boldsymbol{r}_i^W$ and $\boldsymbol{r}_j^C$. Ideally, these representations should be both intra-context-sensitive, i.e. aware of information within the sequences, and inter-sequence-dependent, i.e. utilizing knowledge about the other sequences. These intuitions are reflected in our model design as intra-sequence LSTMs, self-attention and cross-serial attention.

Our model architecture is illustrated in Figure 2. Each word w_i is first mapped to its embedding, and then fed into a bi-directional LSTM (bi-LSTM) that associates each position with a hidden state $\boldsymbol{h}_i^W$. For column headers, since each column name can have multiple words, we apply word embedding lookup and bi-LSTM for each column name, and use the final hidden state from the bi-LSTM as the initial representation for the column name. Next, we apply self-attention (Vaswani et al., 2017) to contextualize this initial representation into $\boldsymbol{h}_j^C$. After obtaining these intra-context-sensitive representations $\boldsymbol{h}_i^W$ and $\boldsymbol{h}_j^C$, we use cross-serial dot-product attention (Luong et al., 2015) to get a weighted average of $\boldsymbol{h}_j^C$ as the context vector for each w_i, and vice versa for each c_j. The two vectors are concatenated and fed into final bi-LSTMs for the natural language question and table column names, respectively. The hidden states of these two LSTMs are our desired context-sensitive representations $\boldsymbol{r}_i^W$ and $\boldsymbol{r}_j^C$.

4   NON-DETERMINISTIC ORACLES

Previously, we assumed that each natural language question has a single corresponding SQL query, and each query has a single underlying correct action sequence. However, these assumptions do not hold in practice. One well-observed example is the ordering of the filtering conditions in the WHERE clause. Reordering of those conditions leads to different action sequences. Furthermore, we identify another source of ambiguity in section 4.2, where a question can be expressed by different SQL queries with the same execution results. These queries are equivalent from an end-user perspective.

For both cases, we obtain multiple correct "reference" transition sequences for each training instance and there is no single target policy for our model to mimic during training. To solve this, we draw inspiration from syntactic parsing and define *non-deterministic* oracles (Goldberg & Nivre, 2013) that allow our parser to explore alternative correct action sequences. In contrast, the training mechanism we discussed in Section 3 is called *static* oracles.

We denote the oracle as O that returns a set of correct continuation actions $O(x, a_{<t})$ at time step t. Taking any action from the set can lead to some desired parse among a potentially large set of correct results. The training objective for each instance L_x is defined as:

$$L_x = \sum_{i=1}^{k} \log \sum_{a \in O(x, a_{<i})} P_\theta(a|x, a_{<i}), \tag{1}$$

where $a_{<i}$ denotes the sequence of actions $a_1, \ldots, a_{i-1}$ and $a_i = \arg\max_{a \in O(x, a_{<i})} s_a$, the most confident correct action to take as decided by the parser during training. When O is a *static* oracle, it always contains a single correct action. In that scenario, Equation 1 is reduced to a naïve cross-entropy loss. When O is *non-deterministic*, the parser can be exposed to different correct action sequences and it is no longer forced to conform to a single correct action sequence during training.

4.1   ALLEVIATING THE "ORDER-MATTERS" ISSUE

Training a text-to-SQL parser is known to suffer from the so-called "order-matters" issue. The filtering conditions of the SQL queries do not presume any ordering. However, an incremental parser must linearize queries and thus impose a pre-defined order. A correct prediction that differs from a golden labeling in its ordering of conditions then may not be properly rewarded. Prior work has tackled this issue through reinforcement learning (Zhong et al., 2017) and a modularized *sequence-to-set* solution (Xu et al., 2017). The former lowers optimization stability and increases training time, while the latter complicates model design to capture inter-dependencies among clauses: information about a predicted filtering condition is useful for predicting the next condition.

We leverage non-deterministic oracles to alleviate the "order-matters" issue. Our model combines the advantage of an incremental approach to leverage inter-dependencies among clauses and the modularized approach for higher-quality training signals. Specifically, at intermediate steps for predicting the next filtering condition, we accept all possible continuations, i.e. conditions that have not been predicted yet, regardless of their linearized positions. For the example in Figure 1, in addition to the transition sequence we gave in Section 3.1, our non-deterministic oracles also accept CONDCOL(c_2) as a correct continuation of the second action. If our model predicts this action first, it will continue predicting the second filtering condition before predicting the first.

4.2   EXECUTION-ORIENTED MODELING OF IMPLICIT COLUMN NAME MENTIONS

In preliminary experiments, we observed that a major source of parser errors on the development set is incorrect prediction of implicit column names. Many natural language queries do not explicitly mention the column name of the filtering conditions. For example, the question in Figure 1 does not mention the column name "Name". Similarly, a typical question like "What is the area of Canada?" does not mention the word "country". For human, such implicit references make natural language queries succinct, and the missing information can be easily inferred from context. But for a machine learning model, they pose a huge challenge.

We leverage the non-deterministic oracles to learn the aforementioned implicit column name mentions by accepting the prediction of a special column name, ANYCOL. During execution,

we expand such predictions into disjunction of filtering conditions applied to all columns, simulating the intuition why a human can easily locate a column name without hearing it from the query. For the example in Figure 1, in addition to the action CONDCOL(c_1), we also allow an alternative prediction CONDCOL(ANYCOL). When the latter appears in the query (e.g. ANYCOL="Willis Tower"), we expand it into a disjunctive clause (Rank="Willis Tower" OR Name="Willis Tower" OR ...). With our non-deterministic oracles, when column names can be unambiguously resolved using the filtering values, we accept both ANYCOL and the column name as correct actions during training, allowing our models to predict whichever is easier to learn.

5 EXPERIMENTS

5.1 DATASET AND EVALUATION

In our experiments, we use the default train/dev/test split of the WikiSQL dataset. We evaluate our models trained with both the static oracles and the non-deterministic oracles on the dev and test split. We report both logical form accuracy (i.e., exact match of SQL queries) and execution accuracy (i.e., the ratio of predicted SQL queries that result in the same answer after execution). The execution accuracy is the metric that we aim to optimize.

5.2 IMPLEMENTATION DETAILS

We largely follow the preprocessing steps in prior work of Dong & Lapata (2018). Before the embedding layer, only the tokens which appear at least twice in the training data are retained in the vocabulary, the rest are assigned a special "UNK" token. We use the pre-trained GloVe embeddings (Pennington et al., 2014), and allow them to be fine-tuned during training. Embeddings of size 16 are used for the actions. We further use the type embeddings for the natural language queries and column names following Yu et al. (2018): for each word w_i, we have a discrete feature indicating whether it appears in the column names, and vice versa for c_j. These features are embedded into 4-dimensional vectors and are concatenated with word embeddings before being fed into the bi-LSTMs. The encoding bi-LSTMs have a single hidden layer with size 256 (128 for each direction). The decoder LSTM has two hidden layers each of size 256. All the attention connections adopt the dot-product form as described in Section 3.2.

For the training, we use a batch size of 64 with a dropout rate of 0.3 to help with the regularization. We use Adam optimizer (Kingma & Ba, 2014) with the default initial learning rate of 0.001 for the parameter update. Gradients are clipped at 5.0 to increase stability in training.

5.3 RESULTS

The main results are presented in Table 2. Our model trained with static oracles achieves comparable results with the current state-of-the-art Coarse2Fine (Dong & Lapata, 2018) and MQAN (McCann et al., 2018) models. On top of this strong model, using non-deterministic oracles during training leads to a large improvement of 2.1% in terms of execution accuracy. The significant drop in the logical form accuracy is expected, as it is mainly due to the use of ANYCOL option for the column choice: the resulting SQL query may not match the original annotation.

We further separate the contribution of "order-matters" and ANYCOL for the non-deterministic oracles. When our non-deterministic oracles only address the "order-matters" issue as described in Section 4.1, the model performance stays roughly the same compared with the static-oracle model. We hypothesize that it is because the ordering variation presented in different training instances is already rich enough for a vanilla sequence-to-action model to learn well. Adding ANYCOL to the oracle better captures the implicit column name mentions and has a significant impact on the performance, increasing the execution accuracy from 81.8% to 83.7%.

Our incremental parser uses a greedy strategy for decoding, i.e. picking the highest scoring action predicted by the policy. A natural extension is to expand the search space using beam search decoding. We further incorporate the execution-guided strategy (Wang et al., 2018) along with beam search. The execution-guided decoder avoids generating queries with semantic errors, i.e. runtime

Model	Dev		Test	
	Acc_{lf}	Acc_{ex}	Acc_{lf}	Acc_{ex}
Coarse2Fine (Dong & Lapata, 2018)	72.5	79.0	71.7	78.5
MQAN (McCann et al., 2018)	**76.1**	82.0	75.4	81.4
Our Models				
INCSQL (static oracle)	**76.1**	82.5	**75.5**	81.6
INCSQL (non-det. oracle, "order-matters" only)	75.4	82.2	75.1	81.8
INCSQL (non-det. oracle)	49.9	84.0	49.9	83.7
INCSQL (non-det. oracle) + EG (5)	51.3	**87.2**	51.1	**87.1**

Table 2: Dev and Test accuracy (%) on WikiSQL. Acc_{lf} refers to logical form accuracy and Acc_{ex} refers to execution accuracy. "+ EG (5) " indicates execution-guided decoding with beam size of 5.

Training Oracles	w/o EG	+ EG (1)	+ EG (3)	+ EG (5)
static	81.6	83.5	86.4	86.7
non-determinstic	83.7	86.0	87.1	87.1
Speed (instances per second)	48.3	30.1	8.2	4.4

Table 3: Execution accuracy (%) and decoding speed of our models on the test set of WikiSQL, with varying decoding beam size. The notation "+ EG (k)" is as in Table 2.

errors and empty results. The key insight is that a partially generated output can already be executed using the SQL engine against the database, and the execution results can be used to guide the decoding. The decoder maintains a state for the partial output, which consists of the aggregation operator, selection column and the completed filtering conditions until that stage in decoding. After every action, the execution-guided decoder retains the top-k scoring partial SQL queries free of runtime exceptions and empty output. At final stage, the query with the highest likelihood is chosen. With $k = 5$, the execution-guided decoder on top of our previous best-performing model achieves an execution accuracy of 87.1% on the test set, setting a new state of the art.

We also report the performance of the static oracle model with execution-guided decoding in Table 3. It comes closely to the performance of the non-deterministic oracle model, but requires a larger beam size, which translates to an increase in the decoding time.

5.4 RESULTS ON OTHER NL2SQL DATASETS

To test whether our model can generalize to other datasets, we perform experiments with the ATIS dataset (Price, 1990; Dahl et al., 1994). ATIS has more diverse SQL structures, including queries on multiple tables and nested queries. To be compatible with our task setting, we only retain examples in the ATIS dataset that are free of nested queries, containing only AND operations and no INNER JOIN operators. We perform table joins and create a single table to be included in the input to our models along with the natural language question. The reduced dataset consists of 933 examples, with 714/93/126 examples in the train/dev/test split, respectively.

Our models trained with the static and non-deterministic oracles (without ANYCOL) achieve accuracy of 67.5% and 69.1% on the test set, respectively. The improvement gained from using non-deterministic oracles during training validates our previous hypothesis: ATIS is a much smaller dataset compared with WikiSQL, therefore explicitly addressing "order-matters" helps here. We didn't apply ANYCOL due to the nature of ATIS data.

6 RELATED WORK

WikiSQL, introduced by Zhong et al. (2017), is the first large-scale dataset with annotated pairs of natural language queries and their corresponding SQL forms on a large selection of table schemas. While its coverage of SQL syntax is weaker than previous datasets such as ATIS (Price, 1990; Dahl et al., 1994) and GeoQuery (Zelle & Mooney, 1996), WikiSQL is highly diverse in its questions,

Model	Dev		Test	
	Acc_{lf}	Acc_{ex}	Acc_{lf}	Acc_{ex}
INCSQL (static oracle)	87.1	88.2	65.9	67.5
INCSQL (non-det. oracle, "order-matters" only)	88.1	89.2	68.3	69.1

Table 4: Dev and Test accuracy (%) of the models on the reduced ATIS dataset, where Acc_{lf} refers to logical form accuracy and Acc_{ex} refers to execution accuracy.

table schemas and contents. This makes it an attractive dataset for neural network modeling. Indeed, a large number of recent works have already been evaluated on WikiSQL (Wang et al., 2017; Xu et al., 2017; Huang et al., 2018; Yu et al., 2018; Wang et al., 2018; Dong & Lapata, 2018; McCann et al., 2018).

NL2SQL is a special case of *semantic parsing*. The task of semantic parsing maps natural language to a logical form representing its meaning, and has been studied extensively by the natural language processing community (see Liang 2016 for a survey). The choice of meaning representation is usually task-dependent, including lambda calculus (Wong & Mooney, 2007), lambda dependency-based compositional semantics (Liang, 2013, λ-DCS), and SQL (Zhong et al., 2017). Neural semantic parsing, on the other hand, views semantic parsing as a sequence generation problem. It adapts deep learning models such as those introduced by Sutskever et al. (2014); Bahdanau et al. (2015); Vinyals et al. (2015). Combined with data augmentation (Jia & Liang, 2016; Iyer et al., 2017) or reinforcement learning (Zhong et al., 2017), sequence-to-sequence with attention and copying has already achieved state-of-the-art results on many datasets including WikiSQL.

The meaning representation in semantic parsing usually has strict grammar syntax, as opposed to target sentences in machine translation. Thus, models are often constrained to output syntactically valid results. Dong & Lapata (2016; 2018) propose models that generate tree outputs through hierarchical decoding and models that use sketches to guide decoding, but they do not explicitly deal with grammar constraints. In contrast, Yin & Neubig (2017) and Krishnamurthy et al. (2017) directly utilize grammar productions during decoding.

Training oracles have been extensively studied for the task of syntactic parsing, where incremental approaches are common (Goldberg & Nivre, 2013). For syntactic parsing, due to the more structurally-constrained nature of the task and clearly-defined partial credits for evaluation, dynamic oracles allow the parsers to find optimal subsequent actions even if they are in some sub-optimal parsing states (Goldberg & Nivre, 2012; Goldberg et al., 2014; Cross & Huang, 2016). In comparison, non-deterministic oracles are defined for the optimal parsing states that have potential to reach a perfect terminal state. To the best of our knowledge, our work is the first to explore non-deterministic training oracles for incremental semantic parsing.

7 CONCLUSIONS

In this paper, we introduce a sequence-to-action incremental parsing approach for the NL2SQL task. With the observation that multiple SQL queries can have the same or very similar semantics corresponding to a given natural language question, we propose to use non-deterministic oracles during training. On the WikiSQL dataset, our model trained with the non-deterministic oracles achieves an execution accuracy of 83.7%, which is 2.3% higher than the current state of the art. We also discuss using execution-guided decoding in combination with our model. This leads to a further improvement of 3.4%, achieving a new state-of-the-art 87.1% execution accuracy on the test set.

To the best of our knowledge, our work is the first to use non-deterministic oracles for training incremental semantic parsers. Designing such non-deterministic oracles requires identification of multiple correct transition sequences for a given training instance, and an algorithm that decides the possible continuations for any intermediate state that will lead to one of the desired terminal states. We have shown promising results for WikiSQL and filtered ATIS dataset and it would be interesting to extend our work to other more complex NL2SQL tasks and to other semantic parsing domains.

REFERENCES

Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81, 1995.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

James Cross and Liang Huang. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1–11, 2016.

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

Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33–43, 2016.

Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 731–742, 2018.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, July 2018.

Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics*, pp. 959–976, 2012.

Yoav Goldberg and Joakim Nivre. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414, 2013.

Yoav Goldberg, Francesco Sartorio, and Giorgio Satta. A tabular method for dynamic oracles in transition-based parsing. *Transactions of the Association for Computational Linguistics*, 2:119–130, 2014.

Richard Hillestad, James Bigelow, Anthony Bower, Federico Girosi, Robin Meili, Richard Scoville, and Roger Taylor. Can electronic medical record systems transform health care? potential health benefits, savings, and costs. *Health affairs*, 24(5):1103–1117, 2005.

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

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 963–973, 2017.

Robin Jia and Percy Liang. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 12–22, 2016.

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

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

Ross Levine, Thorsten Beck, and Asli Demirguc-Kunt. A new database on the structure and development of the financial sector. *World Bank Economic Review*, 14(3):597–605, 2001.

Percy Liang. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*, 2013.

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

Percy Liang, Michael Jordan, and Dan Klein. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 590–599, 2011.

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

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018.

Eric W. T. Ngai, Li Xiu, and Dorothy C. K. Chau. Application of data mining techniques in customer relationship management: A literature review and classification. *Expert systems with applications*, 36(2): 2592–2602, 2009.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543, 2014.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pp. 149–157. ACM, 2003.

P. J. Price. Evaluation of spoken language systems: The ATIS domain. In *Proceedings of the Workshop on Speech and Natural Language*, pp. 91–95, 1990.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, pp. 3104–3112, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28*, pp. 2692–2700, 2015.

Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. Pointing out SQL queries from text. Technical Report MSR-TR-2017-45, November 2017. URL https://www.microsoft.com/en-us/research/publication/pointing-sql-queries-text/.

Chenglong Wang, Po-Sen Huang, Oleksandr Polozov, Marc Brockschmidt, and Rishabh Singh. Execution-Guided Neural Program Decoding. In *ICML workshop on Neural Abstract Machines & Program Induction v2 (NAMPI)*, 2018.

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

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

Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *The 55th Annual Meeting of the Association for Computational Linguistics*, July 2017.

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

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

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