SQLNet: GENERATING STRUCTURED QUERIES FROM NATURAL LANGUAGE WITHOUT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Synthesizing SQL queries from natural language is a long-standing open problem and has been attracting considerable interest recently. Toward solving the problem, the de facto approach is to employ a sequence-to-sequence-style model. Such an approach will necessarily require the SQL queries to be serialized. Since the same SQL query may have multiple serializations, training a sequence-to-sequence-style model is sensitive to the choice from the equivalent serializations. This phenomenon is documented as the "order-matters" problem. Existing state-of-the-art approaches rely on reinforcement learning to reward the decoder when it generates any of the equivalent serializations. However, we observe that the improvement from reinforcement learning is limited.

In this paper, we propose a novel approach, i.e., SQLNet, to fundamentally solve this problem by avoiding the sequence-to-sequence structure when the order does not matter. In particular, we employ a sketch-based approach where the sketch contains a dependency graph, so that one prediction can be done by taking into consideration only the previous predictions that it depends on. In addition, we propose a sequence-to-set model as well as the column attention mechanism to synthesize the query based on the sketch. By combining all these novel techniques, we show that SQLNet can outperform the prior art by 9% to 13% on the WikiSQL task.

1 INTRODUCTION

Semantic parsing is a long-standing open question and has many applications. In particular, parsing natural language descriptions into SQL queries recently attracts much interest from both academia (Yaghmazadeh et al., 2017) and industry (Zhong et al., 2017). We refer to this problem as the natural-language-to-SQL problem (NL2SQL). The de facto standard approach to solve this problem is to treat both the natural language description and SQL query as sequences, and train a sequence-to-sequence model (Vinyals et al., 2015b) or its variants (Dong & Lapata, 2016) which can be used as the parser. One issue of such an approach is that different SQL queries may be equivalent to each other due to commutativity and associativity. For example, consider the following two queries:

```
SELECT result                      SELECT result
WHERE score='1-0' AND goal=16      WHERE goal=16 AND score='1-0'
```

The order of the two constraints in the `WHERE` clause does not affect the execution results of the query, but syntactically, these two are considered as different queries. It is well-known that the order of these constraints affects the performance of a sequence-to-sequence-style model (Vinyals et al., 2016), and it is typically hard to find the best ordering. To mitigate this ordering issue, a typical approach that has been applied in many scenarios is to employ reinforcement learning (Zhong et al., 2017; Hu et al., 2017). The basic idea is that, after a standard supervised training procedure, the model is further trained using a policy gradient algorithm. In particular, given an input sequence, the decoder of a sequence-to-sequence model samples an output sequence following the output distribution, and computes the reward based on whether the output is a well-formed query and whether the query will compute the correct results. This reward can be used by the policy gra-

dient algorithm to fine-tune the model. However, the improvement that can be achieved through reinforcement learning is often limited. For example, on a NL2SQL task called WikiSQL (Salesforce, 2017), the state-of-the-art work (Zhong et al., 2017) reports an improvement of only 2% by employing reinforcement learning.

In this work, we propose SQLNet to fundamentally solve this issue by avoiding the sequence-to-sequence structure when the order does not matter. In particular, we employ a sketch-based approach to generate a SQL query from a *sketch*. The sketch aligns naturally to the syntactical structure of a SQL query. A neural network, called SQLNet, is then used to predict the content for each *slot* in the sketch. Our approach can be viewed as a neural network alternative to the traditional sketch-based program synthesis approaches (Alur et al., 2013; Solar-Lezama et al., 2006; Bornholt et al., 2016). Note that the-state-of-the-art neural network SQL synthesis approach (Zhong et al., 2017) also employs a sketch-based approach, although their sketch is more coarse-grained and they employ a sequence-to-sequence structure to fill in the most challenging slot in the sketch.

As discussed above, the most challenging part is to generate the `WHERE` clause. Essentially, the issue with a sequence-to-sequence decoder is that the prediction of the next token depends on all previously generated tokens. However, different constraints may not have a dependency on each other. In our approach, SQLNet employs the sketch to provide the dependency relationship of different slots so that the prediction for each slot is only based on the predictions of other slots that it depends on. To implement this idea, the design of SQLNet introduces two novel constructions: *sequence-to-set* and *column attention*. The first is designed to predict an unordered set of constraints instead of an ordered sequence, and the second is designed to capture the dependency relationship defined in the sketch when predicting.

We evaluate our approach on the WikiSQL dataset (Salesforce, 2017), which is, to the best of our knowledge, the only large scale NL2SQL dataset, and compare with the state-of-the-art approach, Seq2SQL (Zhong et al., 2017). Our approach results in the exact query-match accuracy of 61.5% and the result-match accuracy of 68.3% on the WikiSQL testset. In other word, SQLNet can achieve exact query-match and query-result-match accuracy of 7.5 points and 8.9 points higher than the corresponding metrics of Seq2SQL respectively, yielding the new state-of-the-art on the WikiSQL dataset.

The WikiSQL dataset was originally proposed to ensure that the training set and test set have a disjoint set of tables. In the practical setting, it is more possible that such an NL2SQL solution is deployed where there exists at least one query observed in the training set for the majority of tables. We re-organize the WikiSQL dataset to simulate this case, and evaluate our approach and the baseline approach, Seq2SQL. We observe that in such a case the advantage of SQLNet over Seq2SQL enlarges by 2 points, and the SQLNet model can achieve an execution accuracy of 70.1%.

To summarize, our main contributions in this work are three-fold. First, we propose a novel principled approach to handle the sequence-to-set generation problem. Our approach avoids the "order-matters" problems in a sequence-to-sequence model and thus avoids the necessity to employ a reinforcement learning algorithm, and achieves a better performance than existing sequence-to-sequence-based approach. Second, we propose a novel attention structure called *column attention*, and show that this helps to further boost the performance over a raw sequence-to-set model. Last, we design SQLNet which bypasses the previous state-of-the-art approach by 9 to 13 points on the WikiSQL dataset, and yield the new state-of-the-art on a NL2SQL task.

2 SQL Query Synthesis from Natural Language Questions and Table Schema

In this work, we consider the WikiSQL task proposed in (Zhong et al., 2017; Salesforce, 2017). Different from most previous NL2SQL datasets (Tang & Mooney, 2001; Price, 1990; Li & Jagadish, 2014; Pasupat & Liang, 2015; Yin et al., 2015), the WikiSQL task has several properties that we would like. First, it provides a large-scale dataset so that a neural network can be effectively trained. Second, it employs crowd-sourcing to collect the natural language questions created by human beings, so that it can help to overcome the issue that a well-trained model may overfit to template-synthesized descriptions. Third, the task synthesizes the SQL query based only on the natural language and the table schema without relying on the table's content. This will help to mit-

Table

Player	No.	Nationality	Position	Years in Toronto	School/Club Team
Antonio Lang	21	United States	Guard-Forward	1999-2000	Duke
Voshon Lenard	2	United States	Guard	2002-03	Minnesota
Martin Lewis	32, 44	United States	Guard-Forward	1996-97	Butler CC (KS)
Brad Lohaus	33	United States	Forward-Center	1996	Iowa
Art Long	42	United States	Forward-Center	2002-03	Cincinnati

Question:

Who is the player that wears number 42?

SQL:

SELECT player WHERE no. = 42

Result:

Art Long

Figure 1: An example of the WikiSQL task.

igate the scalability and privacy issue that alternative approaches may suffer when being applied to realistic application scenarios where large scale and sensitive user data is involved. Fourth, the data is split so that the training, dev, and test set do not share tables. This helps to evaluate an approach's capability to generalize to unseen schema.

We now explain the WikiSQL task. In particular, the input contains two parts: a natural language question stating the query for a table, and the schema of the table being queried. The schema of a table contains both the name and the type (i.e., real numbers or strings) of each column. The output is a SQL query which reflects the natural language question with respect to the queried table.

Note that the WikiSQL task considers synthesizing a SQL query with respect to only one table. Thus, in an output SQL query, only the `SELECT` clause and the `WHERE` clause need to be predicted, and the `FROM` clause can be omitted. We present an example in Figure 1.

The WikiSQL task makes further assumptions to make it tractable. First, it assumes that each column name is a meaningful natural language description, so that the synthesis task is tractable from only the natural language question and column names. Second, any token in the output SQL query is either a SQL keyword or a sub-string of the natural language question. For example, when generating a constraint in the `WHERE` clause, e.g., `name='Bob'`, the token `'Bob'` must appear in the natural language question as a sub-string. This assumption is necessary when the content in a database table is not given as an input. Third, each constraint in the `WHERE` clause has the form of `COLUMN OP VALUE`, where `COLUMN` is a column name, `OP` is one of "$<, =, >, \geq, \leq$", and `VALUE` is a substring of the natural language question as explained above.

Albeit these assumptions, the WikiSQL task is still challenging. Zhong et al. (2017) report that the state-of-the-art task-agnostic semantic parsing model (Dong & Lapata, 2016) can achieve an execution accuracy of merely 37%, while the state-of-the-art model for this task can achieve an execution accuracy of around 60%. Given its several properties discussed at the beginning of this section, we think that the WikiSQL task is a more suitable challenging task than others considered previously. We consider building and tackling the SQL synthesis task of more complex queries as important future work.

3 SQLNet

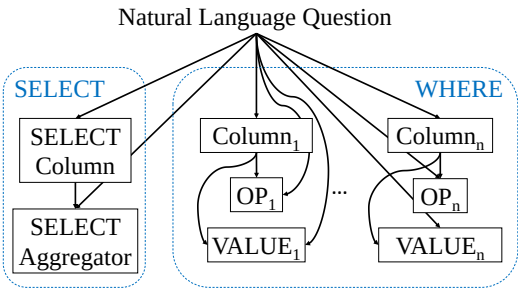
In this section, we present our SQLNet solution to tackle the WikiSQL task. Different with existing semantic parsing models (Dong & Lapata, 2016) which are designed to be agnostic to the output grammar, our basic idea is to employ a *sketch*, which highly aligns with the SQL grammar. Therefore, SQLNet only needs to fill in the *slots* in the sketch rather than predicting both the output grammar and the content.

The sketch is designed to be generic enough, so that all SQL queries of interest can be expressed by the sketch. Therefore, using the sketch does not hinder our approach's generalizability. We will explain the details of a sketch in Section 3.1.

The sketch captures the dependency of the predictions to make. By doing so, the prediction of the value of one slot is only conditioned on the values of those slots that it depends on. This avoids the "order matters" problem in a sequence-to-sequence model, in which one prediction is conditioned on all previous predictions (Vinyals et al., 2016). To make predictions based on a sketch, we de-

SELECT $AGG $COLUMN
WHERE $COLUMN $OP $VALUE
(**AND** $COLUMN $OP $VALUE) *

(a) SQL Sketch



(b) Graphical illustration of the dependency in a sketch

Figure 2: Sketch syntax and the dependency in a sketch

velop two techniques, sequence-to-set and column attention. We will explain the details of these techniques in Section 3.2.

We combine all techniques to design a SQLNet neural network to synthesize a SQL query from a natural language question and a table schema. In Section 3.3, we present the details of SQLNet and training details to surpass previous state-of-the-art approach without using reinforcement learning.

3.1 Sketch-based query synthesis

The SQL sketch that we employ is formally stated in Figure 2a. The tokens in bold (i.e., **SELECT**, **WHERE**, and **AND**) indicate the SQL keywords. The tokens starting with "$" indicate the *slot* to be filled. The name following the "$" indicates the type of the prediction. For example, the `$AGG` slot can be filled with either an empty token or one of the aggregation operators, such as `SUM` and `MAX`. The `$COLUMN` and the `$VALUE` slots need be filled with a column name and a sub-string of the question respectively. The `$OP` slot can take a value from $\{=, <, >\}$. The notion $(...)*$ employ a regular expression to indicate zero or more **AND** clauses.

The dependency graph of the sketch is illustrated in Figure 2b. All slots whose values are to be predicted are illustrated as boxes, and each dependency is depicted as a directed edge. For example, the box of $\texttt{OP}_1$ has two incoming edges from $\texttt{Column}_1$ and the natural language question respectively. These edges indicate that the prediction of the value for $\texttt{OP}_1$ depends on both the values of $\texttt{Column}_1$ and the natural language question. We can view our model as a graphical model based on this dependency graph, and the query synthesis problem as an inference problem on the graph. From this perspective, we can see that the prediction of one constraint is independent with another, and thus our approach can fundamentally avoid the "order-matters" problem in a sequence-to-sequence model.

Note that although it is simple, this sketch is expressive enough to represent all queries in the WikiSQL task. Our SQLNet approach is not limited to this sketch only. To synthesize more complex SQL queries, we can simply employ a sketch that supports a richer syntax. In fact, the state-of-the-art approach on the WikiSQL task, i.e., Seq2SQL (Zhong et al., 2017), can also be viewed as a sketch-based approach. In particular, Seq2SQL predicts for `$AGG` and `$COLUMN` separately from the `WHERE` clause. However, Seq2SQL generates the `WHERE` clause using a sequence-to-sequence model. Thus it still suffers the "order-matters" problem.

3.2 Sequence-to-set prediction using column attention

In this section, we use the prediction of a column name in the `WHERE` clause as an example to explain the ideas of a *sequence-to-set* model and *column attention*. We will explain the full SQLNet model in Section 3.3.

Sequence-to-set. Intuitively, the column names appearing in the `WHERE` clause constitute a subset of the full set of all column names. Therefore, instead of generating a sequence of column names, we can simply predict which column names appear in this subset of interest. We refer to this idea as *sequence-to-set prediction*.

In particular, we compute the probability $P_{\mathbf{wherecol}}(col|Q)$, where col is a column name and Q is the natural language question. To this aim, one idea is to compute $P_{\mathbf{wherecol}}(col|Q)$ as

$$P_{\mathbf{wherecol}}(col|Q) = \sigma(u_c^T E_{col} + u_q^T E_Q) \tag{1}$$

where σ is the sigmoid function, E_{col} and E_Q are the embeddings of the column name and the natural language question respectively, and u_c and u_q are two column vectors of trainable variables. Here, the embeddings E_{col} and E_Q can be computed as the hidden states of a bi-directional LSTM running on top of the sequences of col and Q respectively. Note the two LSTMs to encode the column names and the question do not share their weights. The dimensions of u_c, u_q, E_{col}, E_Q are all d, which is the dimension of the hidden states of the LSTM.

In doing so, the decision of whether or not to include a particular column in the `WHERE` clause can be made independently to other columns by examining $P_{\mathbf{wherecol}}(col|Q)$.

Column attention. Equation (1) has a problem of using E_Q. Since it is computed as the hidden states of the natural language question only, it may not be able to remember the particular information useful in predicting a particular column name. For example, in the question in Figure 1, the token "`number`" is more relevant to predicting the column "`No.`" in the `WHERE` clause. However, the token "`player`" is more relevant to predicting the "`player`" column in the `SELECT` clause. The embedding should reflect the most relevant information in the natural language question when predicting for a particular column.

To incorporate this intuition, we design the *column attention* mechanism to compute $E_{Q|col}$ instead of E_Q. In particular, we assume H_Q is a matrix of $d \times L$, where L is the length of the natural language question. The i-th column of H_Q represents the hidden states output of the LSTM corresponding to the i-th token of the question.

We compute the attention weights w for each token in the question. In particular, w is a L-dimension column vector, which is computed as

$$w = \mathbf{softmax}(v) \qquad v_i = (E_{col})^T W H_Q^i \quad \forall i \in \{1, ..., L\}$$

where v_i indicates the i-th dimension of v, H_Q^i indicates the i-th column of H_Q, and W is a trainable matrix of size $d \times d$.

After the attention weights w are computed, we can compute $E_{Q|col}$ as the weighted sum of each token's LSTM hidden output based on w:

$$E_{Q|col} = H_Q w$$

We can replace E_Q with $E_{Q|col}$ in Equation (1) to get the column attention model:

$$P_{\mathbf{wherecol}}(col|Q) = \sigma(u_c^T E_{col} + u_q^T E_{Q|col}) \tag{2}$$

In fact, we find that adding one more layer of affine transformation before the σ operator can help with the prediction performance by around 1.5%. Thus, we get the final model for predicting column names in the `WHERE` clause:

$$P_{\mathbf{wherecol}}(col|Q) = \sigma((u_a^{col})^T \mathbf{tanh}(U_c^{col} E_{col} + U_q^{col} E_{Q|col})) \tag{3}$$

where U_c^{col} and U_q^{col} are trainable matrices of size $d \times d$, and u_a^{col} is a d-dimensional trainable vector.

We want to highlight that column attention is a special instance of the generic attention mechanism to compute the attention map on a question conditioned on the column names. We will show in our evaluation that this mechanism can improve upon a sequence-to-set model by around 3 points.

3.3 SQLNet MODEL AND TRAINING DETAILS

In this section, we present the full SQLNet model and training details. As illustrated in Figure 2b, the prediction of the `SELECT` clause and `WHERE` clause are separated. In the following, we first present the model for generating the `WHERE` clause and then the `SELECT` clause. In the end, we describe more training details which significantly help to improve the prediction accuracy.

3.3.1 PREDICTING THE WHERE CLAUSE

The WHERE clause is the most complex structure to predict in the WikiSQL task. Our SQLNet model first predicts the set of columns that appear in the WHERE clause based on Section 3.2, and then for each column it generates the constraint by predicting the OP and VALUE slots. We describe them below.

Column slots. After $P_{\mathbf{wherecol}}(col|Q)$ is computed based on Equation (3), SQLNet needs to decide which columns to include in the WHERE. One approach is to set a threshold $\tau \in (0, 1)$, so that all columns with $P_{\mathbf{wherecol}}(col|Q) \geq \tau$ are chosen.

However, we find that an alternative approach can typically give a better performance. We now explain this approach. In particular, we use a network to predict the total number K of columns to be included in the subset, and choose the top-K columns with the highest $P_{\mathbf{wherecol}}(col|Q)$ to form the column names in the WHERE clause.

We observe that most queries have a limited number of columns in their WHERE clauses. Therefore, we set a upper-bound N on the number of columns to choose, and thus we cast the problem to predict the number of columns as a $(N+1)$-way classification problem (from 0 to N). In particular, we have

$$P_{\#\mathbf{col}}(K|Q) = \mathbf{softmax}(U_1^{\#\text{col}} \mathbf{tanh}(U_2^{\#\text{col}} E_{Q|Q}))_i$$

where $U_1^{\#\text{col}}$ and $U_2^{\#\text{col}}$ are trainable matrices of size $(N+1) \times d$ and $d \times d$ respectively. The notion $\mathbf{softmax}(...)_i$ indicates the i-th dimension of the softmax output, and we will use this notion throughout the rest of the description. SQLNet chooses the number of columns K that maximizes $P_{\#\mathbf{col}}(K|Q)$.

In our evaluation, we simply choose $N = 4$ to simplify our evaluation setup. But note that we can get rid of the hyper-parameter N by employing a variant-length prediction model, such as the one for the SELECT column prediction model that will be discussed in Section 3.3.2.

OP slot. For each column in the WHERE clause, predicting the value of its OP slot is a 3-way classifications: the model needs to choose from $\{=, >, <\}$. Therefore, we compute

$$P_{\text{op}}(i|Q, col) = \mathbf{softmax}(U_1^{\text{op}} \mathbf{tanh}(U_c^{op} E_{col} + U_q^{op} E_{Q|col}))_i$$

where col is the column under consideration, $U_1^{\text{op}}, U_c^{op}, U_q^{op}$ are trainable matrices of size $3 \times d$, $d \times d$, and $d \times d$ respectively. Note that $E_{Q|col}$ is used in the right-hand side. This means that SQLNet uses column attention for OP prediction to capture the dependency in Figure 2b.

VALUE slot. For the VALUE slot, we need to predict a substring from the natural language question. To this end, SQLNet employs a sequence-to-sequence structure to generate the sub-string. Note that, here the order of the tokens in the VALUE slot indeed matters. Therefore, using a sequence-to-sequence structure is reasonable.

The encoder phase still employs a bi-directional LSTM. The decoder phase computes the distribution of the next token using a pointer network (Vinyals et al., 2015a; Yang et al., 2016) with the column attention mechanism. In particular, consider the hidden state of the previously generated sequence is h, and the LSTM output for each token in the natural language question is H_Q^i. Then the probability of the next token in VALUE can be computed as

$$P_{\text{val}}(i|Q, col, h) = \mathbf{softmax}(a(h))$$

$$a(h)_i = (u^{\text{val}})^T \mathbf{tanh}(U_1^{\text{val}} H_Q^i + U_2^{\text{val}} E_{col} + U_3^{\text{val}} h) \quad \forall i \in \{1, ..., L\}$$

where u_a^{val} is a d-dimensional trainable vector, $U_h^{\text{val}}, U_c^{\text{val}}, U_q^{\text{val}}$ are three trainable matrices of size $d \times d$, and L is the length of the natural language question. Note that the computation of the $a(h)_i$ is using the column attention mechanism, which is similar in the computation of $E_{Q|col}$.

Note that $P_{\text{val}}(i|Q, col, h)$ represents the probability that the next token to generate is the i-th token in the natural language question.SQLNet simply chooses the most probable one for each step to generate the sequence. Note that the ⟨END⟩ token also appears in the question. The SQLNet model stops generating for VALUE when the ⟨END⟩ token is predicted.

3.3.2 PREDICTING THE SELECT CLAUSE

The SELECT clause has an aggregator and a column name. The prediction of the column name in the SELECT clause is quite similar to the WHERE clause. The main difference is that for the SELECT clause, we only need to select one column among all. Therefore, we compute

$$P_{\mathbf{selcol}}(i|Q) = \mathbf{softmax}(sel)_i$$

$$sel_i = (u_a^{\text{sel}})^T \mathbf{tanh}(U_c^{\text{sel}} E_{col_i} + U_q^{\text{sel}} E_{Q|col_i}) \quad \forall i \in \{1, ..., C\}$$

Here, $u_a^{\text{sel}}, U_c^{\text{sel}}, U_q^{\text{sel}}$ are similar to $u_a^{\text{col}}, U_c^{\text{col}}, U_q^{\text{col}}$ in (3), and C is the total number of columns. Notice that each different dimension of the vector sel is computed based on a corresponding column col_i. The model will predict the column col_i that maximizes $P_{\mathbf{selcol}}(i|Q)$.

For the aggregator, assuming the predicted column name for the SELECT clause is col, we can simply compute

$$P_{\mathbf{agg}}(i|Q, col) = \mathbf{softmax}(U^{\text{agg}} \mathbf{tanh}(U_a E_{Q|col}))_i$$

where U_a is a trainable matrix of size $6 \times d$. Notice that the prediction of the aggregator shares a similar structure as OP.

3.3.3 TRAINING DETAILS

In this section, we present more details to make our experiments reproducible. We also emphasize on the details that can improve our model's performance.

Input encoding model details. Both natural language descriptions and column names are treated as a sequence of tokens. We use the Stanford CoreNLP tokenizer (Manning et al., 2014) to parse the sentence. Each token is represented as a one-hot vector and fed into a word embedding vector before feeding them into the bi-directional LSTM. To this end, we use the GloVe word embedding (Pennington et al., 2014).

Training details. We need a special loss to train the sequence-to-set model. Intuitively, we design the loss to reward the correct prediction while penalizing the wrong prediction. In particular, given a question Q and a set of C columns col, assume y is a C-dimensional vector where $y_j = 1$ indicates that the j-th column appears in the ground truth of the WHERE clause; and $y_j = 0$ otherwise. Then we minimize the following weighted negative log-likelihood loss to train the sub-model for $P_{\mathbf{wherecol}}$:

$$loss(col, Q, y) = -\left(\sum_{j=1}^{C} (\alpha y_j \log P_{\mathbf{wherecol}}(col_j|Q) + (1 - y_j) \log(1 - P_{\mathbf{wherecol}}(col_j|Q)) \right)$$

In this function, the weight α is hyper-parameter to balance the positive data versus negative data. In our evaluation, we choose $\alpha = 3$. For all other sub-module besides $P_{\mathbf{wherecol}}$, we minimize the standard cross-entropy loss.

We choose the size of the hidden states to be 100. We use the Adam optimizer (Kingma & Ba, 2014) with a learning rate 0.001. We train the model for 200 epochs and the batch size is 64. We randomly re-shuffle the training data in each epoch.

Weight sharing details. The model contains multiple LSTMs for predicting different slots in the sketch. In our evaluation, we find that using different LSTM weights for predicting different slots yield better performance than making them share the weights. However, we find that sharing the same word embedding vector helps to improve the performance. Therefore, different components in SQLNet only share the word embedding.

Training the word embedding. In Seq2SQL, Zhong et al. (2017) suggest that the word embedding for tokens appearing in GloVe should be fixed during training. However, we observe that the performance can be boosted by 2 points when we allow the word embedding to be updated during training. Therefore, we initialize the word embedding with GloVe as discussed above, and allow them to be trained during the Adam updates after 100 epochs.

	dev			test		
	Acc_{lf}	Acc_{qm}	Acc_{ex}	Acc_{lf}	Acc_{qm}	Acc_{ex}
Seq2SQL (Zhong et al. (2017))	49.5%	-	60.8%	48.3%	-	59.4%
Seq2SQL (ours)	52.5%	53.5%	62.1%	50.8%	51.6%	60.4%
SQLNet	**-**	**63.2%**	**69.8%**	**-**	**61.3%**	**68.0%**

Table 1: Overall result on the WikiSQL task. Acc_{lf}, Acc_{qm}, and Acc_{ex} indicate the logical form, query-match and the execution accuracy respectively.

4 EVALUATION

In this section, we evaluate SQLNet versus the state-of-the-art approach, i.e., Seq2SQL (Zhong et al., 2017), on the WikiSQL dataset (Salesforce, 2017).

In the following, we first present the evaluation setup. Then we present the comparison between our approach and Seq2SQL on the query synthesis accuracy, as well as a break-down comparison on different sub-tasks. In the end, we propose another variant of the WikiSQL dataset to reflect another application scenario of the SQL query synthesis task, and present our evaluation results of our approach versus Seq2SQL.

4.1 EVALUATION SETUP

In this work, we focus on the WikiSQL dataset (Salesforce, 2017). The dataset was updated on October 16, 2017. In our evaluation, we use the updated version.

We compare our work with Seq2SQL, the state-of-the-art approach on the WikiSQL task. We compare SQLNet with Seq2SQL using three metrics to evaluate the query synthesis accuracy:

1. **Logical-form accuracy.** We directly compare the synthesized SQL query with the ground truth to check whether they match each other. This metric is used in (Zhong et al., 2017).
2. **Query-match accuracy.** We convert the synthesized SQL query and the ground truth into a canonical representation, and compare whether two SQL queries match exactly. This metric can eliminate the false negatives due to only the ordering issue.
3. **Execution accuracy.** We execute both the synthesized query and the ground truth query, and compare whether the results match to each other. This metric is used in (Zhong et al., 2017).

We are also interested in the break-down results on different sub-tasks: (1) the aggregator in the `SELECT` clause; (2) the column in the `SELECT` clause; and (3) the `WHERE` clause. Due to the different structure, it is hard to make further fine-grained comparison.

We implement SQLNet using PyTorch (Facebook, 2017). For the baseline approach in our comparison, i.e., Seq2SQL, we compare our results with the numbers reported by Zhong et al. (2017).

However, Zhong et al. (2017) do not include the break-down results for different sub-tasks, and the source code is not available. To solve this issue, we re-implement Seq2SQL by ourselves. For evaluations whose results are not reported in (Zhong et al., 2017), we report the results from our re-implementation and compare SQLNet against those as the baseline.

4.2 EVALUATION ON THE WIKISQL TASK

Table 1 presents the results for query synthesis accuracy of our approach and Seq2SQL. We first observe that our re-implementation of Seq2SQL yields better result than that reported in (Zhong et al., 2017). Since we do not have access to the source code of the original implementation, we cannot analyze the reason.

We observe that SQLNet outperforms Seq2SQL (even our version) by a large margin. On the logical-form metric, SQLNet outperforms our re-implementation of Seq2SQL by 10.7 points on the dev set and by 10.5 points on the test set. These advancements are even larger to reach 13.7

	dev			test		
	Acc_{agg}	Acc_{sel}	$\text{Acc}_{\text{where}}$	Acc_{agg}	Acc_{sel}	$\text{Acc}_{\text{where}}$
Seq2SQL (ours)	90.0%	89.6%	62.1%	90.1%	88.9%	60.2%
Seq2SQL (ours, C-order)	-	-	63.3%	-	-	61.2%
SQLNet (Seq2set)	-	-	69.1%	-	-	67.1%
SQLNet (Seq2set+CA)	**90.1%**	91.1%	72.1%	**90.3%**	90.4%	70.0%
SQLNet (Seq2set+CA+WE)	**90.1%**	**91.5%**	**74.1%**	**90.3%**	**90.9%**	**71.9%**

Table 2: Break down result on the WikiSQL dataset. Seq2SQL (C-order) indicates that after Seq2SQL generates the `WHERE` clause, we convert both the prediction and the ground truth into a canonical order when being compared. Seq2set indicates that the sequence-to-set technique is employed. +CA indicates that column attention is used. +WE indicates that the word embedding is allowed to be trained. Acc_{agg} and Acc_{sel} indicate the accuracy on the aggregator and column prediction accuracy on the `SELECT` clause, and $\text{Acc}_{\text{where}}$ indicates the accuracy to generate the `WHERE` clause.

points and 13.0 points respectively if we compare with the original results reported in (Zhong et al., 2017). Note that even if we eliminate the false negatives of Seq2SQL by considering the query-match accuracy, the gap is only closed by 1 point, and still remains as large as 9.7 points. We attribute the reason to that Seq2SQL employs a sequence-to-sequence model and thus suffers the "order-matters" problem, while our sequence-to-set-based approach can entirely solve this issue.

On the execution accuracy metric, SQLNet is better than Seq2SQL (reported in Zhong et al. (2017)) by 9.0 points and 8.6 points respectively on the dev and test sets. Although they are still large, the advancements are not as large as those on the query-match metric. This phenomenon shows that, for some of the queries that Seq2SQL cannot predict exactly correct, (e.g., maybe due to the lack of one constraint in the `WHERE` clause), the execution results are still correct. We want to highlight that the execution accuracy is sensitive to the data in the table, which contributes to the difference between query-match accuracy and execution accuracy.

4.3 A BREAK-DOWN ANALYSIS ON THE WIKISQL TASK

We would like to further analyze SQLNet's and Seq2SQL's performance on different sub-tasks as well as the improvement provided by different techniques in SQLNet. The results are presented in Table 2.

We observe that on the `SELECT` clause prediction, the accuracy is around 90%. This shows that the `SELECT` clause is less challenging to predict than the `WHERE` clause. SQLNet's accuracy on the `SELECT` column prediction better than Seq2SQL. We attribute this improvement to the reason that SQLNet employs column attention.

We observe that the biggest advantage of SQLNet over Seq2SQL is on the `WHERE` clause's prediction accuracy. The improvement on the `WHERE` clause prediction is around 11 points to 12 points. Notice that the order of the constraints generated by Seq2SQL matters. To eliminate this effect, we evaluate the accuracy based on a canonical order, i.e., Seq2SQL (ours, C-order), in a similar way as the query-match accuracy. This metric will improve Seq2SQL's accuracy by 1 point, which obeys our observation on the overall query-match accuracy of Seq2SQL. However, we still observe that the SQLNet can outperform Seq2SQL by a large margin. From the break-down analysis, we can observe that the improvements from the usage of a sequence-to-set architecture is the largest to achieve around 6 points. The column attention further improves a sequence-to-set only model by 3 points, while allowing training word embedding gives another 2 points' improvement.

Note that the improvements on the `SELECT` prediction is around 2 points. The improvements from two clauses add up to the 13 points to 14 points improvements in total.

4.4 EVALUATION ON A VARIANT OF THE WIKISQL TASK

In practice, a machine learning model is frequently retrained periodically to reflect the latest dataset. Therefore, it is more often that when a model is trained, the table in the test set is already seen in the

	dev			test		
	Acc_{lf}	Acc_{qm}	Acc_{ex}	Acc_{lf}	Acc_{qm}	Acc_{ex}
Seq2SQL (ours)	54.5%	55.6%	63.8%	54.8%	55.6%	63.9%
SQLNet	**-**	**65.5%**	**71.5%**	**-**	**64.4%**	**70.3%**

Table 3: Overall result on the WikiSQL variant dataset.

training set. The original WikiSQL dataset is split so that the training, dev, and test sets are disjoint in their sets of tables, and thus it does not approximate this application scenario very well.

To better understand different model's performance in this alternative application scenario, we re-shuffle the data, so that all the tables appear at least once in the training set.

On this new dataset, we evaluate both SQLNet and Seq2SQL, and the results are presented in Table 3. We observe that all metrics of both approaches are improved. We attribute this to that all tables in the test set are observed by the models in the training set. This observation meets our expectation. The improvement of SQLNet over Seq2SQL (our implementation) remains the same across different metrics.

5 Related Work

The study of translating natural language to SQL queries has a long history (Warren & Pereira, 1982; Androutsopoulos et al., 1993; 1995; Popescu et al., 2003; 2004; Li et al., 2006; Giordani & Moschitti, 2012; Zhang & Sun, 2013; Li & Jagadish, 2014; Wang et al., 2017). Earlier work focuses on specific databases and requires additional customization to generalize to each new database.

Recent work considers mitigating this issue by incorporating users' guidance (Li & Jagadish, 2014; Iyer et al., 2017). In contrast, SQLNet does not rely on human in the loop. Another direction incorporates the data in the table as an additional input (Pasupat & Liang, 2015; Mou et al., 2016). We argue that such an approach may suffer scalability and privacy issues when handling large scale user databases.

SQLizer (Yaghmazadeh et al., 2017) is a related work handling the same application scenario. SQLizer is also a sketch-based approach so that it is not restricted to any specific database. Different from our work, SQLizer (Yaghmazadeh et al., 2017) relies on an off-the-shelf semantic parser (Berant et al., 2013; Manning et al., 2014) to translate a natural language question into a sketch, and then employs programming language techniques such as type-directed sketch completion and automatic repairing to iteratively refine the sketch into the final query. Since SQLizer does not require database-specific training and its code is not available, it is unclear how SQLizer will perform on the WikiSQL task. In this work, we focus on neural network approaches to handle the NL2SQL tasks.

Seq2SQL (Zhong et al., 2017) is the most relevant work and achieves the state-of-the-art on the WikiSQL task. We use Seq2SQL as the baseline in our work. Our SQLNet approach enjoys all the benefits of Seq2SQL, such as generalizability to unseen schema and overcoming the inefficiency of a sequence-to-sequence model. Our approach improves over Seq2SQL in that by proposing a sequence-to-set-based approach, we eliminate the sequence-to-sequence structure when the order does not matter, so that we do not require reinforcement learning at all. These techniques enable SQLNet to outperform Seq2SQL by 9 points to 13 points.

The problem to parse a natural language to SQL queries can be considered as a special instance to the more generic semantic parsing problem. There have been many works considering parsing a natural language description into a logical form (Zelle & Mooney, 1996; Wong & Mooney, 2007; Zettlemoyer & Collins, 2007; 2012; Artzi & Zettlemoyer, 2011; 2013; Cai & Yates, 2013; Reddy et al., 2014; Liang et al., 2011; Quirk et al., 2015; Chen et al., 2016). Although they are not handling the SQL generation problem, we observe that most of them need to be fine-tuned to the specific domain of interest, and may not generalize.

Dong & Lapata (2016) provide a generic approach, i.e., a sequence-to-tree model, to handle the semantic parsing problem, which yields the state-of-the-art results on many tasks. This approach

is evaluated in (Zhong et al., 2017), and it has been demonstrated less effective than the Seq2SQL approach. Thus, we do not include it in our comparison.

6 CONCLUSION

In this paper, we propose an approach, SQLNet, to handle an NL2SQL task. We observe that all existing approaches employing a sequence-to-sequence model suffer from the "order-matters" problem when the order does not matter. Previous attempts using reinforcement learning to solve this issue bring only a small improvement, e.g., by around 2 points. In our work, SQLNet fundamentally solves the "order-matters" problem by employing a sequence-to-set model to generate SQL queries when order does not matter. We further introduce the column attention mechanism, which can further boost a sequence-to-set model's performance. In total, we observe that our SQLNet system can improve over the prior art, i.e., Seq2SQL, by a large margin ranging from 9 points to 13 points on various metrics. This demonstrates that our approach can effective solve the "order-maters" problem, and shed new light on novel solutions to structural generation problems when order does not matter.

REFERENCES

Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pp. 1–8. IEEE, 2013.

I Androutsopoulos, G Ritchie, and P Thanisch. Masque/sql an efficient and portable natural language query interface for relational databases. *Database technical paper, Department of AI, University of Edinburgh*, 1993.

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

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

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013. ISSN 2307-387X.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 2013.

James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze. Optimizing synthesis with metasketches. 51(1):775–788, 2016.

Qingqing Cai and Alexander Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *ACL*, 2013.

Xinyun Chen, Chang Liu, Richard Shin, Dawn Song, and Mingcheng Chen. Latent attention for if-then program synthesis. *arXiv preprint arXiv:1611.01867*, 2016.

Li Dong and Mirella Lapata. Language to logical form with neural attention. In *ACL*, 2016.

Facebook. Pytorch, 2017. URL `http://pytorch.org/`.

Alessandra Giordani and Alessandro Moschitti. Translating questions to SQL queries with generative parsers discriminatively reranked. In *COLING*, 2012.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. *arXiv preprint arXiv:1704.05526*, 2017.

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

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Fei Li and HV Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.

Yunyao Li, Huahai Yang, and HV Jagadish. Constructing a generic natural language interface for an xml database. In *EDBT*, volume 3896, pp. 737–754. Springer, 2006.

Percy Liang, Michael I Jordan, and Dan Klein. Learning dependency-based compositional semantics. pp. 590–599, 2011.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60, 2014.

Lili Mou, Zhengdong Lu, Hang Li, and Zhi Jin. Coupling distributed and symbolic execution for natural language queries. *arXiv preprint arXiv:1612.02741*, 2016.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *CoRR*, abs/1508.00305, 2015.

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 (EMNLP)*, 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.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, pp. 141. Association for Computational Linguistics, 2004.

P. J. Price. Evaluation of spoken language systems: the atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, pp. 91–95, 1990.

Chris Quirk, Raymond J Mooney, and Michel Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *ACL (1)*, pp. 878–888, 2015.

Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392, 2014.

Salesforce. Wikisql, 2017. URL `https://github.com/salesforce/WikiSQL`.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. *ACM SIGOPS Operating Systems Review*, 40(5):404–415, 2006.

Lappoon R. Tang and Raymond J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, pp. 466–477, Freiburg, Germany, 2001.

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

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *NIPS*, 2015b.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 452–466. ACM, 2017.

David HD Warren and Fernando CN Pereira. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122, 1982.

Yuk Wah Wong and Raymond Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 960–967, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: Query synthesis from natural language. In *OOPSLA*, 2017.

Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. Reference-aware language models. *arXiv preprint arXiv:1611.01628*, 2016.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: Learning to query tables with natural language. *arXiv preprint arXiv:1512.00965*, 2015.

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

Luke S. Zettlemoyer and Michael Collins. 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 (EMNLP-CoNLL-2007*, pp. 678–687, 2007.

Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*, 2012.

Sai Zhang and Yuyin Sun. Automatically synthesizing sql queries from input-output examples. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pp. 224–234. IEEE, 2013.

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