What It Takes to Achieve 100% Condition Accuracy on WikiSQL

Semih Yavuz and **Izzeddin Gur** and **Yu Su** and **Xifeng Yan**
Department of Computer Science, University of California, Santa Barbara
{syavuz,izzeddingur,ysu,xyan}@cs.ucsb.com

Abstract

WikiSQL is a newly released dataset for studying the natural language sequence to SQL translation problem. The SQL queries in WikiSQL are simple: Each involves one relation and does not have any join operation. Despite of its simplicity, none of the publicly reported structured query generation models can achieve an accuracy beyond 62%, which is still far from enough for practical use. In this paper, we ask two questions, "Why is the accuracy still low for such simple queries?" and "What does it take to achieve 100% accuracy on WikiSQL?" To limit the scope of our study, we focus on the `WHERE` clause in SQL. The answers will help us gain insights about the directions we should explore in order to further improve the translation accuracy. We will then investigate alternative solutions to realize the potential ceiling performance on WikiSQL. Our proposed solution can reach up to 88.6% condition accuracy on the WikiSQL dataset.

1 Introduction

A large amount of world's data is stored in relational databases, which require users to master structured query languages such as SQL to query data. It might not be convenient for many users who do not have programming background. Towards removing this huge barrier between non-expert users and data, building reliable natural language interfaces to databases has been a long-standing open problem (Woods, 1973; Androutsopoulos et al., 1995; Popescu et al., 2003).

Recently, there is a renewed interest in natural language interfaces to databases due to the advance in deep learning and the new release of large-scale annotated data such as WikiSQL (Zhong et al., 2017). WikiSQL includes a large collection of questions and their corresponding SQL queries. While the queries in WikiSQL are quite simple: Each involves one relation and does not have join operations, none of the publicly reported SQL generation models (Zhong et al., 2017; Xu et al.) can achieve an accuracy beyond 62%, which is far from enough for practical use. It is not clear yet what level of parsing capabilities are needed to achieve high performance, ideally close to 100% accuracy, on this task.

In this paper, we aim to figure out the level of language understanding required to perform well on the WikiSQL task. Specifically, we focus on the `WHERE` clause generation, which is the most challenging part of this task as reported in (Xu et al.): The accuracies for other clauses like `SELECT` and `AGGREGATE` are over 90% whereas the accuracy for `WHERE` is only around 70%. We aim to conduct the following two studies: (i) Understanding the difficulties of the task and (ii) Investigating alternative solutions to realize the potential ceiling performance.

To this end, we first conduct a careful analysis on a subset of the WikiSQL data to identify the main challenges. This analysis leads to two important observations: (i) $\sim$17% of the questions are either too ambiguous (hard) or require external knowledge to answer, (ii) $\sim$68% of the questions can be answered by exact match or simple paraphrasing, however we surprisingly find that the current best system (Xu et al.) can only get less than 80% accuracy on such simple questions. Deeper analysis on the second observation leads to the conclusion that most of the errors in this category are due to wrongly generated condition values as shown by the example in Figure 1. One can resort to soft/hard look-up based approaches over table content as in previous work (Mou et al., 2017; Iyer et al., 2017) or user interaction (which is also applicable to the first observation) to more accurately recognize the entities in questions for better condition value generation.

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 1702–1711
Brussels, Belgium, October 31 - November 4, 2018.

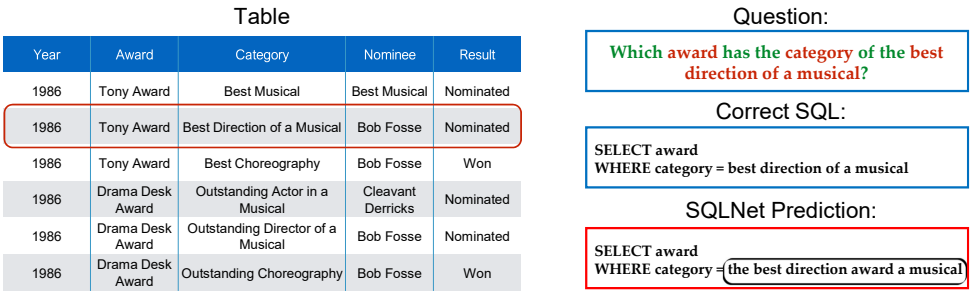

Figure 1: An example from WikiSQL. SQLNet makes a wrong prediction on the condition value of the WHERE clause.

As a second contribution, we propose to use table content as additional data source to address the aforementioned wrongly generated condition value problem, and investigate solutions to realize the potential performance limit (upper bound) on WikiSQL. Note that neither Seq2SQL (Zhong et al., 2017) nor SQLNet (Xu et al.) utilize table content. However, we show that it is not straightforward to achieve a high accuracy even in the scenario where table content is available as an optimal external knowledge through our model ablation results and error analysis. We demonstrate that our proposed solutions can reach up to 88.6% WHERE condition accuracy, almost matching the performance on SELECT and AGGREGATE.

2 Background

The WikiSQL dataset introduced in (Zhong et al., 2017) is created from a large number of tables extracted from Wikipedia, employing Amazon Mechanical Turk for annotation. An example from the dataset is provided in Figure 1: It consists of a table t, a SQL query s, and its corresponding natural language question q. There can be multiple conditions in the WHERE clause of a SQL query, each of which consists of a table column, an operator ($=, <, >$, etc.), and a condition value.

Instead of asking annotators to write SQL queries for given questions and tables, the authors (Zhong et al., 2017) facilitate the annotation process by paraphrasing generated questions. This raises concerns that the resulting dataset is limited to only simple queries. We acknowledge this concern. However, it is still a large, valuable dataset towards the goal of building ultimate natural language interfaces to databases. If the existing or newly proposed solutions can not solve this task with high accuracy, how can we advance to more complicated ones? Any insight and solution to this task can help us build more advanced SQL synthesis algorithms in future.

3 WikiSQL Data Analysis

In this section, we aim to deliver a thorough analysis of the WikiSQL dataset. (Zhong et al., 2017; Xu et al.) made an assumption that only table schema is available to the model, and table content (i.e., table cell values) is not available. We want to answer the following question: As the dataset creation process involves several heuristics and predefined templates to simplify the annotation task, what kind of capabilities does it still require to answer the resulting questions successfully? To this end, we randomly sample 100 examples from the development set of WikiSQL for analysis.

3.1 Categorization

We manually categorize these 100 examples in terms of the capability needed to predict the correct WHERE clause as described below. We also provide illustrative examples for each category in Table 1. If an example belongs to multiple categories, we include it in all the categories that apply. In Table 2, we present the break down of the examples over these categories.

Exact match. For each condition in the SQL query, its column name appears in the question around the neighborhood of its condition value with exactly the same surface form.

Paraphrase. For at least one of the conditions in the SQL query, its column name is paraphrased in the question, hence the inference requires certain paraphrasing capabilities.

Partial clue. For at least one of the conditions in the SQL query, its column name is not explicitly mentioned in the question, not even in paraphrased form. However, there are still partial semantic clues for inference.

External knowledge. For at least one of the conditions in the SQL query, there is no clue in the question to infer its column name. Inferring this column name from the question requires external knowledge regarding the type of its condition

Category	Question	Pseudo SQL Query	Table Columns
EXACT MATCH	In what state was the **electorate** fowler?	SELECT state WHERE **electorate** EQL fowler	member, party, **electorate**, state, term in office
PARAPHRASE	What was the date of the game after week 5 **against the Houston Oilers?**	SELECT date WHERE week GT 5 AND **opponent EQL houston oilers**	week, date, **opponent**, result, attendance
PARTIAL CLUE	Who had the most points in the game **on March 7**?	SELECT high points WHERE **date EQL march 7**	game, **date**, team, score, high points, ...
EXTERNAL-KNOWLEDGE	Name the callback date **for amway arena**	SELECT callback date WHERE **audition venue EQL amway arena**	episode air data, audition city, **audition venue**, callback date, ...
AMBIGUOUS	List the branding **for krca-tv**	SELECT branding WHERE **callsign EQL krca-tv**	branding, **callsign**, channel, power (kw), ...

Table 1: Representative examples for each category. **EQL** and **GT** correspond to the $=$ and $>$ operators. Highlighted parts of the question and the pseudo SQL query are provided to indicate clues for the category.

Category	# Questions	Percentage
EXACT MATCH	59	54.1%
PARAPHRASE	16	14.7%
PARTIAL CLUE	15	13.8%
EXTERNAL KNOWLEDGE	11	10.1%
AMBIGUOUS	8	7.3%

Table 2: Number of examples in different categories.

Category	Seq2set+CA	Seq2set+CA+WE
EXACT MATCH	67.8%	72.9%
PARAPHRASE	75.0%	68.8%
PARTIAL CLUE	80.0%	80.0%
EXTERNAL KNOWLEDGE	54.6%	45.5%
AMBIGUOUS	37.5%	37.5%
TOTAL	67.0%	67.9%

Table 3: Accuracy breakdown of SQLNet.

value that can be detected from the question.

Ambiguous. For this category, even with the external knowledge it is almost impossible (even for humans) to confidently infer the correct condition column from the question.

3.2 Performance Breakdown of SQLNet

In Table 3, we show the performance breakdown of SQLNet (Xu et al.), the state-of-the-art model for WikiSQL, on the selected 100 examples. It has two variants with comparable performance, so we show both of them. As expected, both variants of SQLNet perform poorly on examples that are either too ambiguous or require external knowledge. However, it is surprising that the accuracy on examples that need only exact matching or simple paraphrasing is also not very high, especially considering the paraphrasing capabilities of deep learning models gained from distributed representations. We find that most of these errors are due to wrongly generated condition values as illustrated in Figure 1, where SQLNet fails to even produce a valid phrase. This is indeed important prior knowledge that can be effectively outsourced by resorting to soft/hard look-up based approaches (Mou et al., 2017; Iyer et al., 2017) instead of fully relying on models to precisely generate it.

The above observations exhibit an opportunity to incorporate external knowledge in the condition generation process. We opt to use the table content as the knowledge source to address the wrong condition value problem, which is not used in the existing models (Zhong et al., 2017; Xu et al.). Next we will show that it is not trivial to leverage table content.

4 Our Solutions

As motivated in the previous sections, our main objective here is to investigate solutions to realize the potential ceiling performance on WikiSQL when using the table content as an additional knowledge source. We first describe an attentional RNN-based model that serves as our baseline, and propose several variants for `WHERE` clause generation, each addressing a specific weakness.

4.1 Task Formulation

Given a question q and a table t, our objective is to generate the `WHERE` clause of the SQL query corresponding to the question. In addition, we assume that the table t can be queried while generating the `WHERE` clause. Each condition in the `WHERE` clause is represented as a triple of (`COLUMN`, `OP`, `VALUE`). SQLNet (Xu et al.) generates each of these individual components by first predicting `COLUMN` and `OP`, and then generating `VALUE` using pointer networks (Vinyals et al.). In

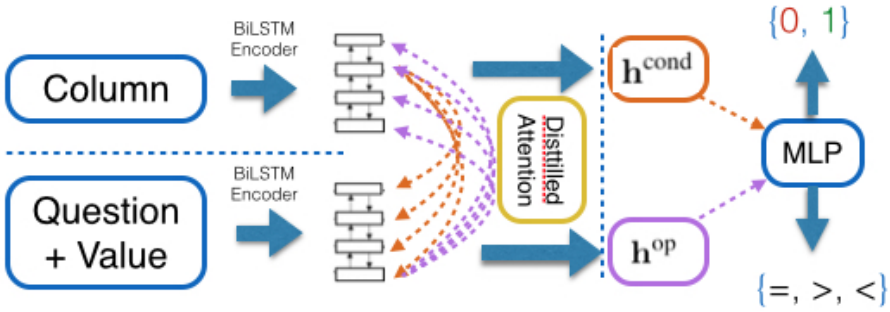

Figure 2: Model overview.

contrast, we propose a two step approach to tackle this problem in the reverse way, taking advantage of the content of table t as additional knowledge: (i) Generate the condition VALUE from the question, and (ii) predict which COLUMN and OP apply to this VALUE.

4.1.1 Candidate Generation

Our objective in candidate generation process is to produce a set of (COLUMN, VALUE) pairs from question q and table t, where VALUE is an *n-gram* in q and COLUMN is a column in t. Similar to prior work (Zhong et al., 2017; Xu et al.), we assume that VALUE occurs in the question.

We first generate the set $\mathcal{N}$ of all *n-grams* from the question. Then, for each candidate $v \in \mathcal{N}$ and each column $c \in t$, we query table t to check whether value v is contained in any row of column c. Then, we create a set $\mathcal{C} = \{(c, v) : v \in c\}$ of (COLUMN, VALUE) pairs as our final candidates for the WHERE clause. We note here that VALUE may not necessarily be an *n-gram* in question for other potential external knowledge or NLIDB tasks. In such scenarios, we can alternatively resort to soft look-up based approaches like the ones proposed in (Mou et al., 2017; Bordes et al., 2015).

4.1.2 Condition Prediction

Given a question q, SQL table t, and a set $\mathcal{C}$ of (COLUMN, VALUE) pair as candidates for WHERE clause, our objective is to learn two mappings:

$$f_{\text{cond}} : (q, c, v) \mapsto \{0, 1\} \quad (1)$$

$$f_{\text{op}} : (q, c, v) \mapsto \{=, >, <\} \quad (2)$$

where f_{cond} determines whether to select $(c, v) \in \mathcal{C}$ as a condition in WHERE clause and f_{op} predicts the operator OP $\in \{=, >, <\}$. These two mappings together can fully determine the final WHERE clause. Note that there can be multiple conditions in the WHERE clause.

4.2 Model Overview

We design a neural network model (Figure 2) to instantiate the mappings f_{cond} and f_{op}. We first describe the general structure of the model, consisting of the following steps:

Value Context. We define *value context* as the context of the value in the question for a given pair of question q and value v. Later the question encoder will condition on this information to make predictions. We will investigate different options for value context in Section 4.3. For now, we define it as a transformation g_{context} which maps each (q, v) to its value context.

Value Abstraction. Inspired by (Yavuz et al., 2016), we define *value abstraction* as a transformation g_{abstract} that replaces the surface form of VALUE in the question with a single token ENTITY. For the running example in Figure 1, applying value abstraction maps the question to "*Which award has the category of the ENTITY?*" It further informs the question encoder regarding the location of the VALUE in the question.

Encoding. Given question $q = (q_1, q_2, \ldots, q_m)$, column $c = (c_1, c_2, \ldots, c_n)$, and value $v = (v_1, v_2, \ldots, v_k)$. Before encoding, we first apply the aforementioned textual transformations and obtain value context

$$q' = (q'_1, q'_2, \ldots, q'_l) = g_{\text{context}} \circ g_{\text{abstract}}(q, v).$$

We first encode all the words into a vector space using an embedding matrix $E \in \mathbb{R}^{d \times |\mathcal{V}|}$, where $\mathcal{V}$ denotes the vocabulary and d is the embedding dimension. Let $\mathbf{q'}_i$ denote the embedding of word q'_i. To obtain the contextual embeddings, we use a bi-directional LSTM with hidden unit size h:

$$\overrightarrow{\mathbf{h}^{\text{q}}}_i = LSTM^{\text{q}}_{\text{fwd}}(\overrightarrow{\mathbf{h}^{\text{q}}}_{i-1}, \mathbf{q'}_i) \quad (3)$$

$$\overleftarrow{\mathbf{h}^{\text{q}}}_i = LSTM^{\text{q}}_{\text{bwd}}(\overleftarrow{\mathbf{h}^{\text{q}}}_{i+1}, \mathbf{q'}_i) \quad (4)$$

with $\overrightarrow{\mathbf{h}}_0 = \mathbf{0}$ and $\overleftarrow{\mathbf{h}}_{l+1} = \mathbf{0}$. We combine forward and backward contextual embed-

dings for each word in the question to obtain $\mathbf{h}_i^{\text{q}} = \mathbf{concat}(\overrightarrow{\mathbf{h}^{\text{q}}}_i, \overleftarrow{\mathbf{h}^{\text{q}}}_i) \in \mathbb{R}^{2h}$.

Similarly, we obtain the contextual embedding $\mathbf{h}_j^{\text{col}} \in \mathbb{R}^{2h}$ for each word c_j in column c with another bi-directional LSTM but shared word embedding matrix.

Distilled Attention. The objective of this step is to distill the most relevant information from both the value context and the column for the final prediction. We first compute the attention score of each word $c_j \in c$ on the words $q'_i \in q'$:

$$S_{j,i}^{(\text{col}\to\text{q})} = \mathbf{h}_j^{\text{col}} \mathbf{W}^{(\text{col}\to\text{q})} \mathbf{h}_i^{\text{q}} \quad (5)$$

$$P_j^{(\text{col}\to\text{q})} = \text{softmax}_i\, S_{j,i}^{(\text{col}\to\text{q})} \quad (6)$$

where $\mathbf{W}^{(\text{col}\to\text{q})} \in \mathbb{R}^{2h\times 2h}$ is a model parameter to be learned and $P_j^{(\text{col}\to\text{q})} \in \mathbb{R}^l$ represents an attention probability distribution of word c_j over the words of value context q'. Let $P^{(\text{col}\to\text{q})} \in \mathbb{R}^{n\times l}$ denote the column-wise concatenation of $P_j^{(\text{col}\to\text{q})}$ indicating the unified representation of attention matrix from column words onto value context. Similarly, we compute the attention from value context (question) to column and get $P^{(\text{q}\to\text{col})} \in \mathbb{R}^{l\times n}$.

The intuition behind *distilled attention* is to allow two-way comparison to clean up the attention weights. To exemplify this point better, consider a scenario where a word $c_j \in c$ attends on a word $q'_i \in q'$ with high probability, but q'_i is much more relevant to another word $c_{j'} \in c$. We leverage reverse attention $P^{(\text{q}\to\text{col})}$ to distill the attention weights of column words on the value context. To this end, we define distilled attention weights as

$$P = (P^{(\text{q}\to\text{col})})^\top \odot P^{(\text{col}\to\text{q})} \quad (7)$$

where $P \in \mathbb{R}^{n\times l}$ becomes our final attention weights and $\odot$ indicates the Hadamard product.

Value Context and Column Representations. Having defined how the encodings and attention weights are computed, we now describe the final value context and column representations. First, we compute a value context vector for each word $c_j \in c$ using the distilled attention weights by

$$\mathbf{h}_j^{(\text{q}\to\text{col})} = \sum_{i=1}^{l} P_{j,i}\, \mathbf{h}_i^{\text{q}} \quad (8)$$

and fuse it with the corresponding column context vector by

$$\mathbf{h}_j^{\text{cond}} = \tanh(\mathbf{W}_0^{\text{cond}} \mathbf{h}_j^{\text{col}} + \mathbf{W}_1^{\text{cond}} \mathbf{h}_j^{(\text{q}\to\text{col})}) \quad (9)$$

$$\mathbf{h}_j^{\text{op}} = \tanh(\mathbf{W}_0^{\text{op}} \mathbf{h}_j^{\text{col}} + \mathbf{W}_1^{\text{op}} \mathbf{h}_j^{(\text{q}\to\text{col})}) \quad (10)$$

where $\mathbf{W}_0^{\text{cond}}, \mathbf{W}_1^{\text{cond}}, \mathbf{W}_0^{\text{op}}, \mathbf{W}_1^{\text{op}} \in \mathbb{R}^{2h\times 2h}$ are trainable model parameters. Finally, we apply a pooling layer to obtain fixed-size representations

$$\mathbf{h}^{\text{cond}} = \frac{1}{n}\sum_j \mathbf{h}_j^{\text{cond}} \text{ and } \mathbf{h}^{\text{op}} = \frac{1}{n}\sum_j \mathbf{h}_j^{\text{op}} \quad (11)$$

for *condition* and *operator* predictions.

Prediction. So far we have obtained two different representations $\mathbf{h}^{\text{cond}} \in \mathbb{R}^{2h}$ and $\mathbf{h}^{\text{op}} \in \mathbb{R}^{2h}$ as the latent unified representations of question, column, and value triple (q, c, v). We than use these representations to make the final predictions:

$$\mathbf{p}^{\text{cond}} = \text{softmax}(\mathbf{U}^{\text{cond}}\, \mathbf{h}^{\text{cond}}) \in \mathbb{R}^2 \quad (12)$$

$$\mathbf{p}^{\text{op}} = \text{softmax}(\mathbf{U}^{\text{op}}\, \mathbf{h}^{\text{op}}) \in \mathbb{R}^3 \quad (13)$$

where $\mathbf{U}^{\text{cond}} \in \mathbb{R}^{2\times 2h}$ and $\mathbf{U}^{\text{op}} \in \mathbb{R}^{3\times 2h}$ are model parameters. The final prediction mappings are then defined as:

$$f_{\text{cond}}(q, c, v) = \arg\max_i \mathbf{p}_i^{\text{cond}} \quad (14)$$

$$f_{\text{op}}(q, c, v) = \arg\max_i \mathbf{p}_i^{\text{op}} \quad (15)$$

Training Objective. Let $\mathcal{T} = \{(q, c, v, l, o)\}$ denote a set of training tuples where $l \in \{0, 1\}$ denotes whether to include a condition with (c, v) as $(\texttt{COLUMN}, \texttt{VALUE})$ pair and if so, $o \in \{=, >, <\}$ indicates which `OP` to apply. Our loss function consists of two components:

$$J(\Theta) = J_{\text{cond}}(\Theta) + l\, J_{\text{op}}(\Theta) \quad (16)$$

where $J_{\text{cond}}(\Theta) = -\log(\mathbf{p}_l^{\text{cond}})$ and $J_{\text{op}}(\Theta) = -\log(\mathbf{p}_o^{\text{op}})$ are simply negative log-likelihood losses for *condition* and *operator* predictions.

Inference. We also employ an inference schema where we make a simple assumption that a condition value v may be a part of *only one* condition. Hence, we group candidate $\{(c, v)\}$ pairs by value v, and create a candidate column set C_v: $\{c\colon (c, v)\}$ for each unique candidate value v. Based on the probabilities $p^{\text{cond}}(c, v)$ computed by the trained model, we select the column $c \in C_v$ with the maximum probability for each candidate value v, hence form the set of $(\texttt{COLUMN}, \texttt{VALUE})$ pairs to be included in condition this way.

4.3 Model Variants

In this section, we discuss variants of our model with different choices for the value context. Consider the running example in Figure 1 and assume

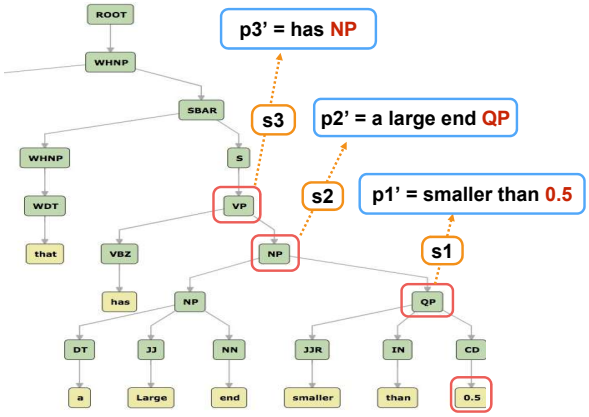

Figure 3: Partial view of parse tree for question "*Which taper/ft that has a Large end smaller than 0.5*" illustrating parse tree based value contexts.

hypothetically that *award* is a candidate condition column for value *best direction of a musical*. When we use the whole question as the value context, eliminating *award* from being a condition COLUMN for this VALUE becomes very challenging for the model as it is not informed enough regarding the finer context of the value.

4.3.1 Base Model

The base model simply uses identity mapping for value context. More precisely, we use the whole question as the context for the candidate condition VALUE. So, $g_{\text{context}}(q, v) = q$.

4.3.2 Window-based model

The objective of *window-based* model is to get value contexts that can provide more clean context information by leveraging the context window around the candidate value. In this case, we first identify the span $[start, end]$ for value v in the question q. Based on its span and a predetermined context window size w, we define $g_{\text{context}}(q, v) = (q_{start-w}, \ldots, q_{start}, \ldots, q_{end}, \ldots, q_{end+w})$.

4.3.3 Parse tree-based model

We also analyze a simple parse tree based model that hierarchically split up the value context into multiple contexts based on the constituency parse tree of the question. To motivate this, consider the question in Figure 3. Window-based value context for 0.5 is *"a large end smaller than ENTITY"* and candidate table columns to apply this value are *"large end"* and *"small end"*. Based on this context, a model will likely assign a higher probability to *"large end"* than *"small end"*. The syntactic structure of the question can potentially help reduce such ambiguities.

Parse tree-based value contexts. As highlighted in Figure 3, we use nested phrase-level subtrees that contain the candidate VALUE. Moreover, to better inform the model regarding the type of phrases, we use phrase level constituent tags of subtrees in two ways: (i) to inform the parent tree with the type of its subtree containing candidate VALUE, (ii) to apply a tag-specific affine transformation on phrase representations. To this end, we first select the r-lowest subtrees $s_1 \subset s_2 \subset \ldots \subset s_r$ of the question's parse tree containing VALUE along with their corresponding phrase-level constituency tags[1] $t_1, t_2, \ldots, t_r$, respectively. We then iteratively form multiple values contexts as nested phrases of $p'_1, p'_2, \ldots, p'_r$ shown in Figure 3 as follows: (i) p'_1 is equal to the phrase formed by the words at the leaf nodes of subtree s_1, and then (ii) iteratively form p'_i as the phrase formed by the words under subtree s_i by replacing its subphrase corresponding to subtree s_{i-1} with its constituency tag t_{i-1} for $i > 1$.

Modifications to General Model. We now describe how the general model defined in Section 4.2 is adapted to accommodate the multiple nested value contexts of different types. In this process, we aim to capture two types of important information: (i) syntactic phrase-level types of value contexts, and (ii) their distance to VALUE.

We first compute a value context vector $\mathbf{h}_{j,k}^{(\text{q}\to\text{col})} \in \mathbb{R}^{2h}$ for each word $c_j \in c$ and each value context p'_k as in Eq. 8 using the same encoding and attention mechanisms. However, we replace the affine transformation layers defined in Eq. 9 and 10 with tag and distance specific ones as follows:

$$\mathbf{h}_j^{\text{cond}} = \tanh(\mathbf{W}_0^{\text{cond}} \mathbf{h}_j^{\text{col}} + \frac{1}{r} \sum_{k=1}^{r} \mathbf{W}_{k,t_k}^{\text{cond}} \mathbf{h}_{j,k}^{(\text{q}\to\text{col})}) \quad (17)$$

$$\mathbf{h}_j^{\text{op}} = \tanh(\mathbf{W}_0^{\text{op}} \mathbf{h}_j^{\text{col}} + \frac{1}{r} \sum_{k=1}^{r} \mathbf{W}_{k,t_k}^{\text{op}} \mathbf{h}_{j,k}^{(\text{q}\to\text{col})}) \quad (18)$$

where $\mathbf{W}_0^{\text{cond}}, \mathbf{W}_{k,t_k}^{\text{cond}}, \mathbf{W}_0^{\text{op}}, \mathbf{W}_{k,t_k}^{\text{op}} \in \mathbb{R}^{d\times 2h}$ for $k = 1, 2, \ldots, r$ are model parameters. It is important to note here that k determines the distance of value context to VALUE and t_k indicates its tag, effectively making the fusion layers above *tag* and

[1] We use the Penn treebank annotation conventions that are described in Bies et al. (1995) at http://languagelog.ldc.upenn.edu/myl/PennTreebank1995.pdf

Model	Dev	Test
No External Knowledge		
SEQ2SQL (Zhong et al., 2017)	62.1%	60.2%
SQLNET-(Seq2Set + CA)	72.1%	70.0%
SQLNET-(Seq2Set + CA + WE)	74.1%	71.9%
SQLNET*-(Seq2Set + CA)	72.3%	70.9%
SQLNET*-(Seq2Set + CA + WE)	73.8%	71.7%
In Our Scenario		
SQLNET*-(Seq2Set + CA)	82.2%	80.9%
SQLNET*-(Seq2Set + CA + WE)	**83.5%**	**81.8%**
STAMP + RL (Sun et al., 2018)	77.3%	76.3%
TYPESQL + TC (w/ Freebase) (Yu et al., 2018)	**92.8%**	**87.9%**
OUR BASELINE	**77.2%**	**77.1%**
SQLMASTER	84.8%	83.9%
SQLMASTER + VA	86.2%	86.1%
SQLMASTER (Window-Based)	87.4%	87.1%
SQLMASTER (Window-Based) + VA	87.9%	87.6%
SQLMASTER (Tree-Based)	88.7%	88.3%
SQLMASTER (Tree-Based) + VA	**88.9%**	**88.6%**
OUR UPPERBOUND	**92.1%**	**91.4%**

Table 4: `WHERE` clause accuracy. +VA denotes that *Value Abstraction* is applied. SQLNET* refers to our reimplementation of SQLNet. SQLMASTER refers to our proposed models. Only TYPESQL + TC leverages Freebase on top of table content among the models reported in our scenario. WHERE clause accuracy of TYPESQL (w/o Freebase) is not reported.

distance specific. The rest of the model exactly follows the Eq. from 11 through 15 with the same training objective and inference schemes.

5 Experiments

In this section, we discuss the details of the experiments and present our main results.

5.1 Training Details

For training our neural networks, we only keep the words appearing at least 3 times in the whole training data and the rest of the words are replaced with `UNK` token. Word embeddings are initialized with pretrained GloVe (Pennington et al., 2014) vectors[2], and updated during the training. We take the dimension of word embeddings and the size of LSTM hidden layer both equal to 100. The model parameters are optimized using Adam (Kingma and Ba, 2015) with batch size of 32 and a decaying learning rate starting with 0.001. We apply gradient clipping to 5 when its norm exceeds this value. We use early stopping based on the model accuracy on the development set. We report our results with a model snapshot achieving the best accuracy on the development set. Our models are implemented in *tensorflow* (Abadi et al., 2016). The code is available at `https://github.com/semihyavuzz/sql_master`.

[2]More specifically, we use 100D vectors from http://nlp.stanford.edu/data/glove.6B.zip

Model	Dev	Test
SQLNET (Xu et al.)	63.2%	61.3%
DIALSQL (Gur et al., 2018)	70.9%	69.0%
TYPESQL (w/o Freebase) (Yu et al., 2018)	66.5%	64.9%
TYPESQL + TC (w/ Freebase) (Yu et al., 2018)	**79.2%**	**75.4%**
SQLMASTER (Ours, w/o Freebase)	**73.1%**	**72.4%**

Table 5: Full query-match (QM) accuracy. For SQLMASTER, we combine our WHERE clause predictions with the SELECT and AGGREGATE clause predictions of SQLNet. QM result for TYPESQL + TC (w/o Freebase) is not reported.

5.2 Main Results

In Table 4, we present our main results in comparison with the related works. BASELINE refers to a baseline for our models where the `WHERE` clause accuracy is computed by assuming that each candidate (`COLUMN`, `VALUE`) pair is included with corresponding `OP` being equality. On the other hand, UPPERBOUND accuracy is computed by assuming f_{cond} and f_{op} makes 100% correct mapping of whether to include a candidate (`COLUMN`, `VALUE`) pair and which `OP` to apply on this condition. In other words, errors in UPPERBOUND exist due to wrong candidate generation.

As shown in Table 4, our models surpass the previous results by a large margin as well as its variants improving upon each other. A portion of these improvements definitely come from assuming and using the table itself as the optimal external knowledge. Acknowledging this fact, we make the following more important conclusions from Table 4, 5, and 6: (i) Comparison of the performance results across scenarios (SQLNET vs. SQLMASTER or TYPESQL) reveals that there is a large room for improvement when an external knowledge base is used, (ii) Comparison of our own models with its variants demonstrates that each component/extension incorporated brings a considerable performance improvement, justifying its potential power to be used in other related NLP tasks, (iii) Comparing our models with SQLNET and TYPESQL+TC within our scenario provides further clues/justifications towards effectiveness of our proposed Tree-Based model, (iv) SQLMASTER performs comparably to TYPESQL + TC (Yu et al., 2018) on WHERE condition predictions despite the fact that we do not use Freebase, which is exploited in (Yu et al., 2018) to identify named entities of certain Freebase types (e.g., *person*, *place*, *country*, *organization*, *sport*), (v) Our SQLMASTER (Tree-Based) + VA model achieves 88.6% on test portion of WikiSQL, which almost reaches the upper bound of 91.4%, demonstrating a great promise for future work in this do-

Category	SQLNET	SQLMASTER
EXACT MATCH	72.9%	86.4%
PARAPHRASE	68.8%	93.8%
PARTIAL CLUE	80.0%	93.3%
EXTERNAL KNOWLEDGE	45.5%	90.9%
AMBIGUOUS	37.5%	75.0%
TOTAL	67.9%	88.1%

Table 6: Accuracy breakdown of SQLNET compared to SQLMASTER over the categories obtained by hand-analysis on the randomly selected examples as explained in Section 3.

main, and finally (vi) When the performance of our model is compared to SQLNET over the categories as shown in Table 6, we observe that it consistently improves the performance of over all the categories, but most noticeably on EXTERNAL KNOWLEDGE and AMBIGUOUS ones which were the main categories inspiring this work and proposed approaches motivated by the analysis provided in Section 3.

Evaluation in Our Scenario. We adapt SQLNet (Xu et al.) results to our scenario via a post-processing step as follows: For each of their predicted condition in `WHERE` clause, if column c and operator o are both correct, but value v is wrong, then we replace this value with the one in our generated candidates that maps to the column c when the mapping is unique (only one value maps to c).

5.3 Error Analysis

In this section, we provide an error analysis of our models to better understand what are the remaining challenges to achieve UPPERBOUND performance. To this end, we analyzed 100 randomly sampled examples from development set on which our best model fails. 41% of these errors are due to our models not being able to perform a good semantic understanding of the value context. 36% of the errors correspond to ambiguous questions that lack sufficient information to disambiguate between correct and wrong column. A good representative example for this category is "*Who was the director of king's speech?*", where our model predicts "*winner and nominees*" column for the condition value "*king's speech*" while the correct column is "*original title*". The remaining 18% and 5% of the errors are caused by sparsity of column names and wrong labelling problems, respectively.

6 Related Work

Research on natural language interfaces to databases (NLIDBs) and semantic parsing has spanned several decades. Early rule-based NLIDBs (Woods, 1973; Androutsopoulos et al., 1995; Popescu et al., 2003) employ carefully designed rules to map natural language questions to formal representations like SQL queries. While having a high precision, rule-based systems are brittle when facing with language variations and usually only admit inputs from a restricted vocabulary. The rise of statistical models (Zettlemoyer and Collins, 2005; Kate et al., 2005; Berant et al., 2013) and neural network models (Yih et al., 2015; Dong and Lapata, 2016; Sun et al., 2016; Zhong et al., 2017; Xu et al.) has enabled NLIDBs that are more robust to language variations. Such systems allow users to formulate questions with greater flexibility instead of having to probe and adapt to the boundary of rule-based systems.

Along with the advance in modeling is the development of benchmarks for training and testing NLIDB models. Early benchmarks are mostly curated by experts (Zelle and Ray, 1996; Zettlemoyer and Collins, 2007). State-of-the-art models (Dong and Lapata, 2016) have achieved a high accuracy of 80% to 90% on these benchmarks. In the recent years, a number of large-scale, crowd-sourced benchmarks have been constructed with the goal to train and test NLIDBs in a more real-world setting, notably WebQuestions (Berant et al., 2013) and GraphQuestions (Su et al., 2016) for knowledge bases, and WikiSQL (Zhong et al., 2017) for SQL queries to relational databases. The best accuracies on these benchmarks are still far from enough for real use, typically in the range of 20% to 60%.

Besides releasing WikiSQL, Zhong et al. (2017) propose an approach (Seq2SQL) to solve this task. Seq2SQL leverages the pointer-networks (Vinyals et al.) to generate linearized SQL queries token-by-token using the input question and table schema. They report significant performance improvement over (Dong and Lapata, 2016), a generic sequence-to-tree approach proposed for semantic parsing. More recently, Xu et al. propose a sketch-based sequence-to-set approach (SQLNet) eliminating sequence-to-sequence structure employed in (Zhong et al., 2017), when the order does not matter. In our work, we provide a careful analysis of SQLNet results to better understand the limitations of this model on the WikiSQL task. Inspired by this analysis, we propose novel solutions to realize close to upper-bound condition accuracy in the scenario where SQL table is available as an

optimal external knowledge. Another recent work (Yu et al., 2018) also focuses on using external knowledge (Freebase) along with the table content to generate SQL queries in a type aware fashion. A concurrent line of related work exploits graph-to-sequence neural models with the aim to better exploit syntactic information in the input question (Xu et al., 2018a,b). On the other hand, Gur et al. (2018) takes an orthogonal approach and introduces a dialogue-based query refinement mechanism where a candidate SQL query (generated by any black-box model) is refined by interactively validating and updating modular segments of the query with users. The authors show that by having successful interactions with users, not only the accuracy of the candidate queries can be improved but also new insights into limitations of current query generation systems can be gained.

There are also a number of recent studies on semantic parsing for semi-structured tables. For example, Pasupat and Liang (2015) develop the WikiTableQuestions benchmark, where the task is to find table cells in HTML table to answer questions, while Jauhar et al. (2016) focus on multi-choice questions. On the other hand, Sun et al. (2016) study how to answer user questions with table cells from millions of HTML tables. These studies directly find cells of semi-structured tables as answers, instead of generating SQL queries for relational databases.

7 Conclusion

In this paper, we thoroughly analyzed the recently released WikiSQL dataset and the performance breakdown of SQLNet. Through the analysis, we identified an opportunity/need to further explore the potentials of incorporating external knowledge in the structured query generation process. In this direction, we developed alternative solutions to explore the potential performance limits for this task in the scenario where table content can be used. We showed that our proposed systems can reach up to 88.6% accuracy in condition generation and provided a discussion regarding what the remaining challenges were through an error analysis. We consider solving the WikiSQL task as a necessary preliminary step towards realizing natural language interfaces to databases in full fledge.

Acknowledgements

This research was sponsored in part by the Army Research Laboratory under cooperative agreements W911NF09-2-0053 and NSF 1528175. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.

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

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods on Natural Language Processing (EMNLP)*.

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for treebank ii style penn treebank project. In *Linguistic Data Consortium*.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. Dialsql: Dialogue based structured query generation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

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

Sujay Kumar Jauhar, Peter Turney, and Eduard Hovy. 2016. Tables as semi-structured knowledge for question answering. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Rohit J Kate, Yuk Wah Wong, and Raymond J Mooney. 2005. Learning to transform natural to formal languages. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Lili Mou, Zhengdong Lu, Hang Li, and Zhi Jin. 2017. Coupling distributed and symbolic execution for natural language queries. In *International Conference on Machine Learning (ICML)*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods on Natural Language Processing (EMNLP)*.

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

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for qa evaluation. In *Empirical Methods on Natural Language Processing (EMNLP)*.

Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *World Wide Web (WWW)*.

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

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems (NIPS)*.

William A Woods. 1973. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the American Federation of Information Processing Societies Conference*.

Kun Xu, Lingfei Wu, Zhiguo Wang, and Vadim Sheinin. 2018a. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*.

Kun Xu, Lingfei Wu, Zhiguo Wang, Mo Yu, Liwei Chen, and Vadim Sheinin. 2018b. Exploiting rich syntactic information for semantic parsing with graph-to-sequence model. In *Empirical Methods on Natural Language Processing (EMNLP)*.

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

Semih Yavuz, Izzeddin Gur, Yu Su, Mudhakar Srivatsa, and Xifeng Yan. 2016. Improving semantic parsing via answer type inference. In *Empirical Methods on Natural Language Processing (EMNLP)*.

Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

John M Zelle and Mooney Ray. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

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