Leveraging Table Content for Zero-shot Text-to-SQL with Meta-Learning

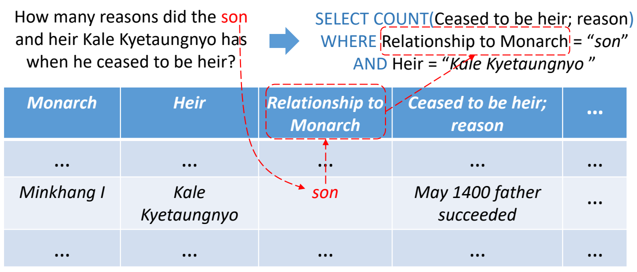
Abstract

Single-table text-to-SQL aims to transform a natural language question into a SQL query according to one single table. Recent work has made promising progress on this task by pre-trained language models and a multi-submodule framework. However, zero-shot table, that is, the invisible table in the training set, is currently the most critical bottleneck restricting the application of existing approaches to real-world scenarios. Although some work has utilized auxiliary tasks to help handle zero-shot tables, expensive extra manual annotation limits their practicality. In this paper, we propose a new approach for the zero-shot text-to-SQL task which does not rely on any additional manual annotations. Our approach consists of two parts. First, we propose a new model that leverages the abundant information of table content to help establish the mapping between questions and zero-shot tables. Further, we propose a simple but efficient meta-learning strategy to train our model. The strategy utilizes the two-step gradient update to force the model to learn a generalization ability towards zero-shot tables. We conduct extensive experiments on a public open-domain text-to-SQL dataset WikiSQL and a self-made domain-specific dataset ElectricitySQL. Compared to existing approaches using the pre-trained model of the same level, our approach achieves significant improvements on both datasets. Compared to the approaches using larger pre-trained models, our approach is still competitive. More importantly, on the zero-shot subsets of both the datasets, our approach further increases the improvements.

Introduction

Since the release of WikiSQL (Zhong, Xiong, and Socher 2017), a large-scale text-to-SQL benchmark, single-table text-to-SQL task has become an active research area in recent years. The goal of the task is to transform natural language questions into Structured Query Language (SQL) to query a single table. Although the search space is limited to one table, the task still has a considerable number of application scenarios (e.g., query regional electricity prices or flight schedules). More importantly, it is the basis for more complex text-to-SQL tasks on multi-tables (Yu et al. 2018c). Therefore, the research on this area is of great significance.

Relying on large-scale pre-trained language models (Devlin et al. 2019) and a multi-submodule framework, existing



How many reasons did the son and heir Kale Kyetaungnyo has when he ceased to be heir?

```
SELECT COUNT(Ceased to be heir; reason)
WHERE Relationship to Monarch = "son"
AND Heir = "Kale Kyetaungnyo"
```

Monarch	Heir	Relationship to Monarch	Ceased to be heir; reason	...
...	...	...	...	...
Minkhang I	Kale Kyetaungnyo	son	May 1400 father succeeded	...
...	...	...	...	...

Figure 1: An example of table content to help predict headers. Red indicates the matching.

approaches (He et al. 2019; Hwang et al. 2019; Lyu et al. 2020) have made considerable progress on the single-table text-to-SQL task. However, few of them pay attention to the challenge of zero-shot tables whose *schema* are not visible in the training set. Typically, in comparison with the visible tables, zero-shot tables are more challenging because they are not directly involved in training. Their schema cannot be perceived by the model, so that they may be noisy in the test. In fact, with the rapid expansion of business, zero-shot tables are becoming more and more common in realistic scenarios. Therefore, in order to make text-to-SQL land from laboratory to application, it is necessary to make the model learn to handle zero-shot tables.

(Chang et al. 2020) explicitly deals with zero-shot tables for the first time. The core idea of their approach is to design an auxiliary task to model the mapping from the question to the headers (similar to entity linking). However, this approach requires training data to provide the gold mappings that are annotated manually. Undoubtedly, it is a strong limitation in realistic scenarios.

In this paper, we propose a new approach called *Meta-Content text-to-SQL* (MC-SQL) to handle zero-shot tables. The motivation comes from the following two intuitions: 1) The first one is that table content can provide abundant information for predicting headers. Figure 1 shows an example. The cell *son* in the table is relevant to the question word "son", thus reveals the potential header *Relationship to Monarch*. 2) The second one is that meta-learning can help the model learn the generalization ability between different tables from the training data. It is because meta-learning

```
    SELECT  $AGG  $SEL
(WHERE  $COL  $OP  $VAL)
 (AND  $COL  $OP  $VAL)*
```

Figure 2: Skeleton of SQL in single-table text-to-SQL.

has the capability that only needs a few gradient steps to quickly adapt to new tasks. Specifically, our approach consists of two parts. On the one hand, a table content-enhanced model is employed to encode questions, headers, and table cells at the same time, in order to combine their semantic relevance for the prediction on zero-shot tables. On the other hand, a zero-shot meta-learning algorithm is utilized to train our content-enhanced model instead of the traditional mini-batch strategy. In each training step, the algorithm generalizes the model by two sets of samples that rely on two disjoint table sets, respectively. Finally, to comprehensively evaluate our approach, we conduct experiments on public open-domain benchmark WikiSQL and domain-specific benchmark ElectricitySQL built by ourselves. The contribution of this paper is summarized below.

- We propose a new text-to-SQL model that leverages table content to enhance the header prediction on zero-shot tables. To the best of our knowledge, it is the first attempt to explicitly utilize table content to handle zero-shot tables.
- We leverage meta-learning to force the model to learn the generalization capability from the different tables of the training set, in order to adapt to zero-shot tables.
- We conduct comprehensive experiments on two text-to-SQL benchmarks. Our approach achieves a significant improvement over the baselines with the same-level pre-trained model, and also achieves competitive results over the baselines with the larger-level pre-trained model.

Preliminaries

The single-table text-to-SQL task can be formally defined as

$$y = \mathcal{M}(q, \mathcal{T}) \tag{1}$$

where q denotes a natural language question and y denotes the corresponding SQL query. $\mathcal{T} = \{h^1, h^2, ..., h^l\}$ denotes the table which q relies on, where h^i denotes the i-th header in $\mathcal{T}$. The goal of the task is to learn a mapping $\mathcal{M}$ from questions to SQL queries. In addition, this task supposes that no complex SQL syntax (e.g., GROUP BY and nested query) exists and there is only one column in the SELECT clause. Specifically, each y follows a unified skeleton, which is shown in Figure 2. The tokens prefixed with "$" indicate the slots to be filled and "*" indicates zero or more AND clauses. According to the skeleton, existing approaches (He et al. 2019; Hwang et al. 2019; Lyu et al. 2020) break the total task into the following six subtasks:

- **Select-Column(SC)** finds the column $SEL in the SELECT clause from $\mathcal{T}$.
- **Select-Aggregation(SA)** finds the aggregation function $AGG ($\in$ {NONE, MAX, MIN, COUNT, SUM, AVG}) of the column in the SELECT clause.

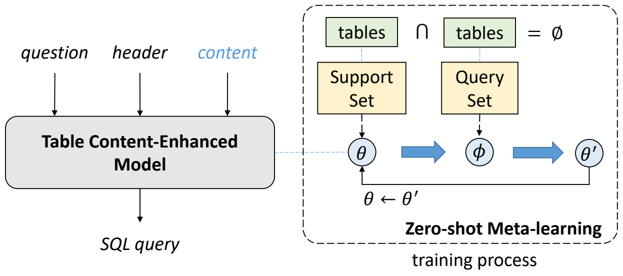


Figure 3: Overall framework of our approach.

- **Where-Number(WN)** finds the number of where conditions, denoted by $\mathcal{N}$.
- **Where-Column(WC)** finds the column (header) $COL of each WHERE condition from $\mathcal{T}$.
- **Where-Operator(WO)** finds the operator $OP ($\in \{=, >, <\}$) of each $COL in the WHERE clause.
- **Where-Value(WV)** finds the value $VAL for each condition from the question, specifically, locating the starting position of the value in q.

There are dependencies between some tasks. For example, the prediction of $OP requires $COL, and the prediction of $VAL requires both $COL and $OP.

Approach

The framework of our approach is shown in Figure 3, which consists of two parts. First, the *table content enhanced model* (left) captures the semantic relevance of questions with headers and cells at the same time, and predict subtasks comprehensively. Further, *zero-shot meta-learning* (right) is leveraged to train the table content enhanced model. In each training batch, the model parameters are updated in two stages to force the model to learn the generalization ability.

Table Content Enhanced Model

Figure 1 demonstrates that the core of leveraging table content is to find the table cells mentioned in the question. However, the total number of the cells can be very large, far exceeding that of the headers. Consequently, it is impractical to directly embed all of them.

To overcome this challenge, we adopt coarse-grained filtering before embedding. Specifically, for each header h, only the cell c with the highest literal similarity to question q will be retained. The literal similarity is computed by

$$\varphi(c; q) = \max_{\mathrm{n}(q)} \frac{\mathrm{lcs}(\mathrm{n}(q), c)}{2|\mathrm{n}(q)|} + \frac{\mathrm{lcs}(\mathrm{n}(q), c)}{2|c|} \tag{2}$$

where $\mathrm{n}(q)$ denotes the n-gram of q, $|x|$ denotes the length of string x, and $\mathrm{lcs}(x, y)$ denotes the length of the *Longest Consecutive Common Subsequence* between the string x and y. The intuitive meaning of $\varphi(c; q) \in [0, 1]$ is that the larger proportion of the overlap in the two strings, the higher the similarity of them. In addition, if the retained cell whose score is smaller than the threshold σ, it will be replaced with

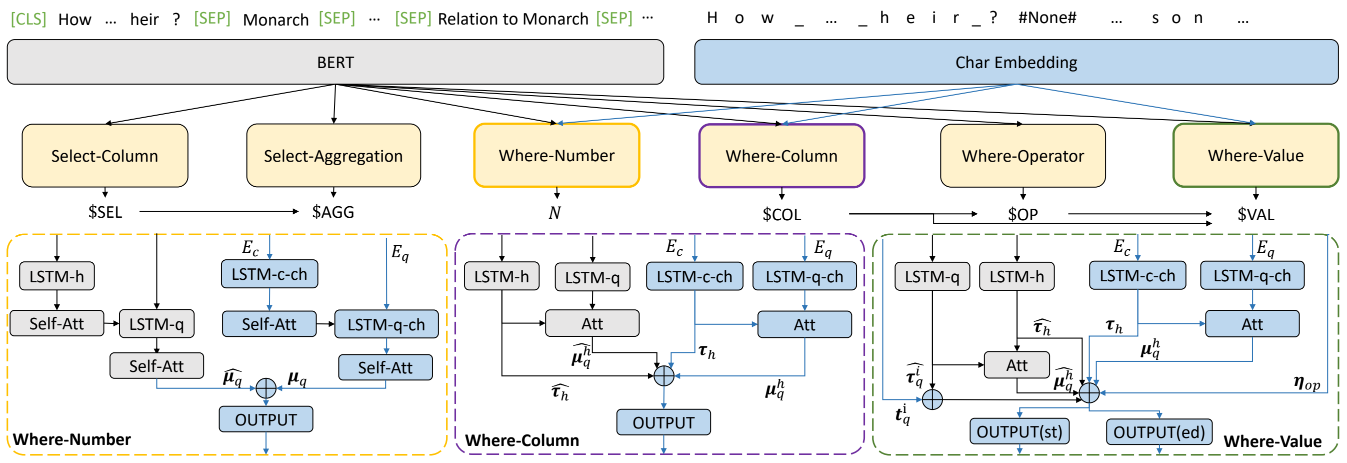


Figure 4: Architecture of the table content-enhanced model. WN, WC, and WV are detailed in the orange, purple, and green dotted box, respectively. Blue indicates the processes for table content and gray indicates the processes for headers.

a special token #None#, in order to avoid noise. After filtering, each header has a corresponding cell (or #None#).

The overall architecture of our table content-enhanced model is shown in Figure 4. It consists of an encoding module and six sub-modules corresponding to six sub-tasks. Intuitively, table content is mainly helpful to the sub-tasks for the WHERE clause, especially WN, WC, and WV. Therefore, we will detail these three sub-modules that utilize table content.

Encoding Module The encoding module consists of BERT (Devlin et al. 2019) and an embedding layer. BERT is employed to encode the question and the headers. Following the format of BERT, the input is a sequence of tokens, which starts with a special token `[CLS]` followed by the word tokens of question q and all headers $\{h^i\}$. A special token `[SEP]` is utilized to separate the q and each h^i. For each input token, BERT outputs a hidden vector that holds its context information. In addition to BERT, the embedding layer is utilized to embed the cells. Differing from BERT, the embedding is for characters rather than words. It is because cells are typically some entity names and numerical values, etc. Char-embedding can reduce the number of Out-Of-Vocabulary (OOV) tokens. Specifically, For each cell c, its character embedding is denoted by $E_c \in \mathbf{R}^{m\times d_e}$, where m is the character number of c and d_e is the dimension of each embedding. Since q and c should be embedded in the same vector space for calculating their semantic relevance, we also embed q instead of directly using the BERT output of q. The char-embedding of q is denoted by $E_q \in \mathbf{R}^{n\times d_e}$.

Where-Number Sub-Module This sub-module contains two similar processes, which calculate the header-aware and content-aware context vector of the question, respectively. Here detail the process of the latter. First, for each h, its content vector $\boldsymbol{\tau}_h \in \mathbf{R}^d$ is obtained by a BiLSTM on E_c followed by a max-pooling operation ($\mathbf{R}^{m\times d} \to \mathbf{R}^d$). Then, the hidden vectors of q is obtained by the other BiLSTM, denoted by $\Gamma_q \in \mathbf{R}^{n\times d}$. In order to make this BiLSTM aware of c when encoding q, its initial states $\boldsymbol{s} \in \mathbf{R}^{2d}$ is obtained by performing a self-attention on all $\boldsymbol{\tau}_h$.

$$\alpha_h = \frac{\exp(\boldsymbol{\tau}_h W_\alpha)}{\sum_{h'\in H}\exp(\boldsymbol{\tau}_{h'} W_\alpha)} \tag{3}$$

$$\boldsymbol{\mu}_h = \sum_{h\in H}^{n} \alpha_h \boldsymbol{\tau}_h \tag{4}$$

$$\boldsymbol{s} = W_s \mu_h \tag{5}$$

Here, $\alpha_h \in \mathbf{R}$ is the attention weight of each header h. $W_\alpha \in \mathbf{R}^{d\times d}$ and $W_s \in \mathbf{R}^{d\times 2d}$ are the trainable parameter matrices. Thereafter, the content-aware question context $\boldsymbol{\mu}_q \in \mathbf{R}^d$ is calculated by a self-attention on Γ_q, which is similar to $\boldsymbol{\mu}_h$. As described at the beginning, the header-aware question context $\widehat{\boldsymbol{\mu}}_q \in \mathbf{R}^d$ is calculated by the same procedures above on the output of BERT[1]. Finally, the result $\mathcal{N}$ is predicted by combining the header- and content-aware context vectors.

$$\mathcal{N} = \arg\max_i p_{wn}(i|q,\mathcal{T}) \tag{6}$$

$$p_{wn}(i|q,\mathcal{T}) = \tanh([\widehat{\boldsymbol{\mu}}_q;\boldsymbol{\mu}_q]W_\mu)W_o \tag{7}$$

where $W_\mu \in \mathbf{R}^{2d\times d}$ and $W_o \in \mathbf{R}^{d\times 1}$ are the trainable parameter matrices.

Where-Column Sub-Module In this sub-module, the processes of calculating question hidden vectors Γ_q and content vectors $\boldsymbol{\tau}_h$ are similar to those in WN. The only difference is the initial states of all the BiLSTMs are all random. Thereafter, in order to make the model focus on the parts of q that are relevant to c, an attention mechanism is performed on Γ_q to calculate content-aware context vector $\boldsymbol{\mu}_q^h \in \mathbf{R}^d$.

$$\boldsymbol{\mu}_q^h = \sum_{i=1}^{n} \alpha_q^i \gamma_q^i \tag{8}$$

[1]In order to distinguish easily, all the vectors obtained from the output of BERT are marked with a hat, such as $\widehat{x}$.

$$\alpha_q^i = \frac{\exp(\boldsymbol{\gamma}_q^i W_\alpha \boldsymbol{\tau}_h{}^T)}{\sum_{j=1}^n \exp(\boldsymbol{\gamma}_q^j W_\alpha \boldsymbol{\tau}_h{}^T)} \tag{9}$$

where $\boldsymbol{\gamma}_q^i \in \mathbf{R}^d$ is the i-th hidden vector in Γ_q and $\alpha_q^i \in \mathbf{R}$ is its attention weight. $W_\alpha \in \mathbf{R}^{d \times d}$ is the parameter matrix. Finally, the result \$COL is predicted by

$$\$\mathrm{COL} = \arg\max_{h \in \mathcal{T}} p_{wc}(h|q, \mathcal{T}) \tag{10}$$

$$p_{wc}(h|q, \mathcal{T}) = \tanh([\widehat{\boldsymbol{\mu}}_q^h; \widehat{\boldsymbol{\tau}}_h; \boldsymbol{\mu}_q^h; \boldsymbol{\tau}_h])W_o \tag{11}$$

where $\widehat{\boldsymbol{\tau}}_h$ and $\widehat{\boldsymbol{\mu}}_q^h$ are the header vector and header-aware context vector, respectively. They are obtained by decoding the output of BERT with the same steps towards $\boldsymbol{\mu}_q^h$ and $\boldsymbol{\tau}_h$.

Where-Value Sub-Module The architecture of this sub-module is almost consistent with that of WC. The difference is that prediction of \$VAL also requires the results of WC and WV, namely \$COL and \$OP. Let h denote \$COL and op denote \$OP, then the starting position st and ending position ed of \$VAL can be calculated by

$$st = \arg\max_{w^i \in q} p_{st}(w^i|q, \mathcal{T}) \tag{12}$$

$$ed = \arg\max_{w^i \in q} p_{ed}(w^i|q, \mathcal{T}) \tag{13}$$

$$p_{st}(w^i|q, \mathcal{T}) = \tanh([\widehat{\boldsymbol{\mu}}_q^h; \widehat{\boldsymbol{\tau}}_h; \boldsymbol{\mu}_q^h; \boldsymbol{\tau}_h; \boldsymbol{\eta}_{op}; \boldsymbol{\xi}_q^i])W_{st} \tag{14}$$

$$p_{ed}(w^i|q, \mathcal{T}) = \tanh([\widehat{\boldsymbol{\mu}}_q^h; \widehat{\boldsymbol{\tau}}_h; \boldsymbol{\mu}_q^h; \boldsymbol{\tau}_h; \boldsymbol{\eta}_{op}; \boldsymbol{\xi}_q^i])W_{ed} \tag{15}$$

Where w^i is the i-th word token of q. $\widehat{\boldsymbol{\mu}}_q^h$, $\widehat{\boldsymbol{\tau}}_h$, $\boldsymbol{\mu}_q^h$, and $\boldsymbol{\tau}_h$ are calculated by the same procedures used in WC. $\boldsymbol{\eta}_{op}$ is the one-hot vector of \$OP and $\boldsymbol{\xi}_q^i$ is the semantic vector of w^i. Here, in order to leverage the content more directly, we propose a Value Linking(VL) strategy for calculating $\boldsymbol{\xi}_q^i$.

$$\boldsymbol{\xi}_q^i = [\widehat{\boldsymbol{\tau}}_q^i; \boldsymbol{t}_q^i] \tag{16}$$

where $\widehat{\boldsymbol{\tau}}_q^i \in \mathbf{R}^d$ is the hidden vector of w^i. It is calculated by a BiLSTM encoding the q output of BERT. $\boldsymbol{t}_q^i \in \mathbf{R}^{d_t}$ denotes the type embedding of w^i. There are only two types, denoted by `Match` and `NotMatch`, that indicate whether w^i matches some cells c, respectively. Initially, the type of each w^i is labeled as `NotMatch`. When calculating the literal similarity by (2), if a cell c is select, all the words of the corresponding n-gram $n(q)$ will be labeled as `Match`.

The architectures of the remaining three modules SC, SA, and WO are almost consistent with WC, except that they do not need the process for table content (i.e., removing the blue process in Figure 4). All their results are predicted by the classification that depends on the combined context $[\widehat{\boldsymbol{\tau}}_h; \widehat{\boldsymbol{\mu}}_q^h]$.

Zero-Shot Meta Learning Framework

Meta-learning is typically leveraged to deal with classification problems and has the ability to adapt quickly between different categories. In our proposed framework, the table that each sample (question) relies on is regarded as an abstract category, in order to create the conditions for applying meta-learning. Furthermore, the traditional meta-learning framework consists of two stages of meta-training and meta-test. However, it is already demonstrated in (Vinyals et al. 2016; Snell, Swersky, and Zemel 2017) where without fine-tuning on the meta-test, the meta-learning model shows similar even better performance. Motivated by this, our meta-learning algorithm only retains the meta-training stage. The entire process is formally described in Algorithm 1.

Algorithm 1 Zero-Shot Meta-Learning Framework

```
Require: A set of training samples $\mathcal{D} = \{(q^i, \mathcal{T}^i, y^i)\}$,
    where $q^i$ is the $i$-th input question, $t^i$ is the table which
    $q^i$ relies on, and $y^i$ is the gold SQL query of $q^i$. A model
    $\mathcal{M}(q, \theta)$, where $\theta$ is its parameters. Hyperparameters $\alpha$,
    $\beta$ and $\gamma$
 1: while not done do
 2:     for all task do
 3:         Sample a support set $\mathcal{S} = \{(q^j, \mathcal{T}^j, y^j)\} \subseteq \mathcal{D}$
 4:         Evaluate $\nabla_\theta \mathcal{L}_\mathcal{S} = \nabla_\theta \Sigma_j \mathcal{L}(\mathcal{M}(q^j, \mathcal{T}^j, \theta), y^j)$
 5:         Update parameters with gradient descent:
            $\phi = \theta - \alpha \nabla_\theta \mathcal{L}_\mathcal{S}$
 6:         Sample a query set $\mathcal{Q} = \{(q^k, \mathcal{T}^k, y^k)\} \subseteq \mathcal{D}$,
            where $\{\mathcal{T}^j\} \cap \{\mathcal{T}^k\} = \emptyset$
 7:         Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{S}\leftarrow\mathcal{Q}} = \nabla_\theta \Sigma_k \mathcal{L}(\mathcal{M}(q^k, \mathcal{T}^k, \phi), y^k)$
 8:         Update $\theta$ to minimum $\mathcal{L}$ using Adam optimizer
            with learning rate $\beta$,
            where $\mathcal{L} = \gamma \mathcal{L}_\mathcal{S} + (1 - \gamma)\mathcal{L}_{\mathcal{S}\leftarrow\mathcal{Q}}$
 9:     end for
10: end while
```

A task consists of several training samples. It is the basic training unit of our framework and split into a support set and a query set. Here, to simulate the scenario of zero-shot tables, the table set of the support set is disjoint with that of the query set. According to the split, the model experienced a two-stage gradient update during the training of each task. In the first stage, temporary parameters ϕ are obtained by calculating the loss of the support set $\mathcal{S}$ and perform the gradient updating on original parameters θ. In the second stage, the loss of the query set $\mathcal{Q}$ is first calculated with ϕ. Then, the losses of the support set and query set are jointed to calculate the gradient. Finally, original parameters θ are updated by the gradient. In addition, for sampling $\mathcal{S}$ and $\mathcal{Q}$, we follow the N-way K-shot setting, i.e, each set covers N tables and there are K samples for each table.

A key difference with the Model-Agnostic Meta-Learning (MAML) algorithm (Finn, Abbeel, and Levine 2017) is that the table sets (classes) of $\mathcal{S}$ and $\mathcal{Q}$ are identical in MAML, whereas, they are disjoint in our algorithm. Following our setting, the model needs to find out the potential common ground between two different table sets and carry out the joint optimization. Therefore, it can force the model to learn the generalization ability on different tables. In addition, in order to reduce the time cost brought by the second-order gradient in MAML, we use the first-order accumulation loss instead to calculate the joint loss (Line 8 in Algorithm 1).

Experiments

Experimental Setup

Our models are trained and evaluated over the following two text-to-SQL benchmarks:

WikiSQL (Zhong, Xiong, and Socher 2017) is a open-domain text-to-SQL benchmark, containing more than 20K tables. Each question corresponds to a table, which is extracted from the Wikipedia page. The data set is divided into 56,355 training questions, 8,421 development questions, and 15,878 test questions[2]. (Chang et al. 2020) demonstrates that 70% of tables in the development and test set have the same schema as those in the training set. Therefore, in order to focus on evaluating the performance on zero-shot tables, we also conduct experiments on the remaining 30% of tables (zero-shot subset) released by (Chang et al. 2020)[3].

ElectricitySQL is a domain-specific text-to-SQL dataset built by ourself. Its format imitates WikiSQL, containing 17 tables. These tables are related to the field of electric energy, including information such as electricity sales, electricity prices, etc. Although the number of tables in ElectricitySQL is small, the number of headers in each table is several times that in a WikiSQL table, thus still covers a wealth of information. In addition, all the tables of ElectricitySQL are extracted from realistic scenarios thus have practical significance. The data set is divided into 10,000 training questions, 1,000 development questions, and 2,000 test questions. In order to simulate the challenge of zero-shot tables, the training set contains only 10 tables of all, while the development set and the test set contain all the tables. We respectively extract the questions from the development and test set that rely on the remaining 7 tables as the zero-shot subsets.

Following previous approaches (Zhong, Xiong, and Socher 2017; Hwang et al. 2019), we adopt logical form (LF) accuracy and execution (EX) accuracy as the evaluation metrics. Here, LF evaluates the literal accuracy of the total SQL query and its clauses, and EX evaluates the accuracy of the results by executing the SQL query.

Implementation Details We perform all the experiments on NVIDIA Tesla V100 GPU. In the experiments, all the BERT models are of *base* version. The following hyperparameters are tuned on development sets: (1) Fitlering threshold σ is set to 0.9 for both datasets. (2) The layer number of all BiLSTMs is set to 2. (3) The hidden state size d is set to 100. (4) The character embedding size d_e is set to 128. (5) The type embedding size d_t is set to 32. (6) The number of sampling tasks is set to 10,000 for WikiSQL, 2,500 for ElectricitySQL. (7) For WikiSQL, both N and K in the N-way K-shot setting are set to 4. For ElectricitySQL, they are set to 1 and 4, respectively. (8) γ in Algorithm 1 is set to 0.3 for WikiSQL, 0.5 for ElectricitySQL. (9) For α in Algorithm 1, BERT and sub-modules are trained with two kinds respectively. Specifically, α_{BERT} is set to 1×10^{-5} and α_{sub} is set to 1×10^{-3}. Similarly, β_{BERT} is set to 1×10^{-5} and β_{sub} is set to 1×10^{-3}.

Overall Results on WikiSQL

We first compared our approach with several existing text-to-SQL approaches on public benchmark WikiSQL. Seq2SQL (Zhong, Xiong, and Socher 2017), Coarse2Fine (Dong and Lapata 2018), and Auxiliary Mapping (Chang et al. 2020) are all sequence-to-sequence (Seq2Seq) based models. SQLova (Hwang et al. 2019) replaces the Seq2Seq framework with the multi-submodule framework for the first time. X-SQL (He et al. 2019) and HydratNet (Lyu et al. 2020) improve this framework by MT-DNN (Liu et al. 2019) and a pair-wise ranking mechanism respectively, thus achieve better results on WikiSQL. Here, we ignore all the results of using the execution guiding (EG) trick (Wang et al. 2018). It is because that EG works on the premise that the generated SQL query must not be empty, which is unreasonable in realistic scenarios.

Approach	Dev LF	Dev EX	Test LF	Test EX
Seq2SQL	49.5	60.8	48.3	59.4
Coarse2Fine	72.5	79.0	71.7	78.5
Auxiliary Mapping	76.0	82.3	75.0	81.7
SQLova(-)	80.3	85.8	79.4	85.2
SQLova(*)	81.6	87.2	80.7	86.2
X-SQL(*)	83.8	89.5	83.3	88.7
HydratNet(*)	83.6	89.1	**83.8**	89.2
MC-SQL(-)	**84.1**	**89.7**	83.7	**89.4**

Table 1: Overall results on WikiSQL. "x(-)" denotes the model x with *base*-BERT. "x(*)" denotes the model x with *large*-BERT or larger pre-trained model, such as MT-DNN (Liu et al. 2019) in X-SQL. Under such harsh conditions, our approach still achieves competitive results.

The overall experimental results on WikiSQL are reported in Table 1. All the results of the compared approaches come from their original papers. On LF accuracy, our approach achieves state-of-the-art results on the development set, and ranks second only to HydratNet (-0.1%) on the test set. On EX accuracy, our approach achieves state-of-the-art results on both the sets. Notably, our results are achieved by only utilizing the *base* version of BERT. After ignoring the baselines that use larger pre-trained models ("(*)" in Table 1), our approach achieves significant improvements on both LF (4.3%) and EX (4.2%) accuracy when testing.

The performance of Seq2SQL, Coarse2Fine, and Auxiliary Mapping is limited by the decoding without SQL syntax constraints. SQLova achieves the significant improvements due to the prior knowledge of BERT and its multi-submodule framework that is more detailed than the syntax-free decoding. Although X-SQL and HydratNet achieve further improvements by stronger language models and more fine-grained ranking strategies respectively, they all ignore the abundant information from table content, thus their performance is also limited. On the contrary, our model makes full use of table content, and further leverages meta-learning to force the model to learn the generalization capability, thus achieving better results.

Detailed Analysis

Ablation Test To explore the contributions of various components of our MC-SQL model, we compared the following settings on both the datasets.

[2]https://github.com/salesforce/WikiSQL

[3]https://github.com/JD-AI-Research-Silicon-Valley/auxiliary-task-for-text-to-sql

Setting	SC	SA	WN	WC	WO	WV	LF
SQLova	96.7 / 96.3	90.1 / 90.3	98.4 / 98.2	94.1 / 93.6	97.1 / 96.8	94.8 / 94.3	80.2 / 79.7
MC-SQL	96.9 / 96.4	**90.5** / 90.6	**99.1 / 98.8**	97.9 / **97.8**	**97.5 / 97.8**	**96.7 / 96.9**	**84.1 / 83.7**
w/o TC	**97.0** / 96.5	89.8 / 90.0	98.6 / 98.3	94.5 / 93.7	97.2 / 97.0	94.7 / 94.7	79.9 / 79.2
w/o VL	**97.0 / 96.7**	90.4 / **90.8**	99.0 / 98.7	**98.0** / 97.6	**97.5** / 97.2	95.6 / 95.5	82.9 / 83.0
w/o ML	96.5 / 96.2	90.4 / 90.4	98.9 / 98.7	97.8 / 97.4	**97.5** / 97.4	96.5 / 96.1	83.2 / 82.9
SQLova	98.4 / 97.8	97.3 / 97.5	99.4 / 99.5	79.2 / 79.3	97.8 / 96.8	86.6 / 86.9	65.0 / 65.3
MC-SQL	**99.9 / 98.8**	97.3 / 97.6	**100.0 / 100.0**	89.0 / **89.4**	99.3 / **99.5**	**88.9 / 88.8**	**75.7 / 75.3**
w/o TC	97.8 / 98.3	**97.5 / 97.9**	99.9 / **100.0**	79.2 / 79.5	98.7 / 98.7	87.3 / 87.5	65.7 / 66.5
w/o VL	98.0 / 98.0	97.3 / 97.6	99.9 / **100.0**	**89.5** / 89.3	**99.5** / 99.3	88.0 / 88.5	75.1 / 75.0
w/o ML	98.2 / 98.0	97.3 / 97.6	99.9 / **100.0**	88.4 / 88.9	**99.5** / 99.3	88.6 / 88.6	73.8 / 74.0

Table 2: Results of sub-tasks on WikiSQL and ElectricitySQL. x/y denotes the results of the dev/test sets.

Setting	SC	SA	WN	WC	WO	WV	LF
SQLova	95.8 / 95.2	89.7 / 89.3	97.6 / 97.4	91.1 / 90.4	95.9 / 95.7	90.1 / 90.5	74.7 / 72.8
MC-SQL	**96.4** / 95.5	**91.1 / 91.0**	**98.7** / 98.1	**96.6 / 96.3**	**97.1 / 96.7**	**94.8 / 94.2**	**82.4 / 80.5**
w/o TC	96.2 / **95.7**	91.0 / 90.5	97.6 / 97.7	91.5 / 90.7	96.2 / 96.1	90.5 / 90.8	75.8 / 73.6
w/o VL	96.2 / 95.8	90.6 / 90.9	98.7 / 98.0	97.1 / **96.3**	97.1 / 96.3	91.7 / 92.1	79.0 / 79.1
w/o ML	95.7 / 95.0	90.4 / 90.2	98.5 / **98.2**	96.0 / 95.8	96.8 / **96.7**	94.0 / 93.5	81.2 / 79.4
SQLova	98.4 / 97.6	95.0 / 95.1	99.5 / 99.2	72.1 / 70.1	98.2 / 97.9	85.8 / 84.3	59.3 / 56.9
MC-SQL	**99.1 / 99.3**	95.3 / 95.6	**100.0 / 100.0**	87.9 / **87.6**	99.6 / **99.6**	**87.9 / 87.3**	**74.3 / 72.4**
w/o TC	**99.1** / 98.8	**95.4 / 95.7**	99.8 / **100.0**	72.9 / 70.4	99.3 / 99.0	**87.9** / 87.0	61.5 / 58.9
w/o VL	98.2 / 98.1	95.3 / 95.5	**100.0 / 100.0**	**88.4** / 87.5	**99.8 / 99.6**	86.9 / 86.7	73.3 / 72.3
w/o ML	98.4 / 98.2	95.1 / 95.1	99.8 / **100.0**	86.8 / 87.2	99.5 / 99.5	87.0 / 86.9	71.2 / 71.5

Table 3: Results of zero-shot subsets on WikiSQL and ElectricitySQL. x/y denotes the results of the dev/test sets.

- **w/o table content(TC)** We removed all the processes in WN, WC, and WV that related to table content. For example, (7) is converted to $p_{wn}(i|q, \mathcal{T}) = \tanh(\widehat{\boldsymbol{\mu}}_q W_\mu) W_o$.
- **w/o value linking(VL)** We retained the processes related to TC but removed the value linking in WV, i.e., (16) is converted to $\boldsymbol{\xi}_q^i = \widehat{\boldsymbol{\tau}}_q^i$ after removing.
- **w/o meta-learning(ML)** We replaced the meta-learning strategy with the traditional mini-batch strategy.

The results of the ablation test on WikiSQL and ElectricitySQL are shown in Table 2, respectively. The upper and bottom blocks show the results of WikiSQL and ElectricitySQL, respectively. MC-SQL equipped with all the components achieves the optimal results on LF accuracy and most subtasks, which significantly improves the strong baseline SQLova on both WikiSQL (4.0%) and ElectricitySQL (10.0%). By removing TC, the overall performance (LF) declined approximately 3.5% and 10% on both the datasets. It demonstrates the significance of the table content. Here, the performance drop by removing VL also proves the contribution of value linking. Removing ML brings a certain drop on both the datasets, however, the drop on ElectricitySQL (-1.3%) is more significant than that on WikiSQL (-0.8%). The reason for the difference can be that WikiSQL is an open-domain dataset, thus it is more difficult for learning the generalization capability than on domain-specific ElectricitySQL.

By further observation, it can be found that the contribution of TC is mainly reflected in the four subtasks of WN, WC, WO, and WV. The improvement on WO is mainly attributed to the improvement on WC, because the former depends on the result of the latter. In addition, meta-learning is helpful for all subtasks and has the most significant improvements on the subtasks that are not enhanced by table content, such as SC and SA. Interestingly, the performance sometimes becomes better on WC and WO after removing VL, which reveals that VL can be noisy for predicting $COL. However, due to its significant improvement on SQL, VL is still helpful for the overall results.

Zero-shot Test We evaluated our model on the zero-shot subsets of both the datasets. The results are shown in Table 3. In terms of overall results, MC-SQL achieves greater improvements over SQLova on the zero-shot subsets of both WikiSQL (7.7% vs 4.0%) and ElectricitySQL (15.5% vs 10.0%). It proves that our approach is promising for handling zero-shot tables. Furthermore, the improvement on each subtask is also increased, especially WN, WC, and WV. It reveals that the key to handling zero-shot tables is the prediction of where conditions. The contribution of table content is greater in zero-shot tables, which is consistent with our intuition. The relatively more drastic performance drop caused by removing ML also proves that our meta-learning strategy is suitable for dealing with zero-shot tables. Notably, in addition to SC and SA, meta-learning is also contributing to the WHERE clause when handling zero-shot tables. It is interesting that the performance on ElectricitySQL is generally lower than that on WikiSQL, whereas the im-

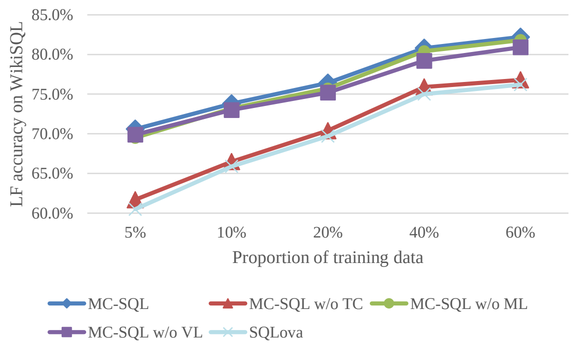


Figure 5: LF accuracy on WikiSQL with proportions of training data.

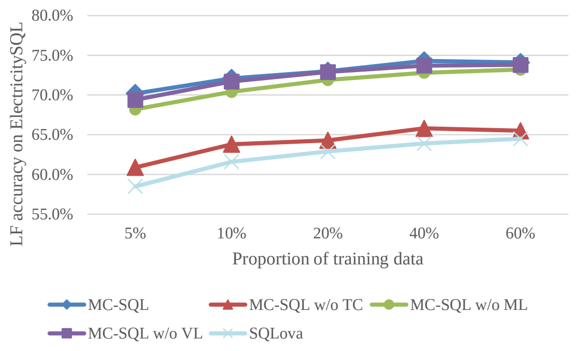


Figure 6: LF accuracy on ElectricitySQL with proportions of training data.

provement brought by meta-learning on ElectricitySQL is greater than that of WikiSQL. We speculate that it is because fewer training tables result in poor performance, and meta-learning, which has the characteristic of suitable for a few samples, achieves greater improvements.

Varied Sizes of Training Data To simulate the scenario of zero-shot tables from another aspect, we tested the performance of the model using different proportions of training data. The results of WikiSQL and ElectricitySQL are shown in Figure 5 and Figure 6, respectively. The MC-SQL equipped with all components always maintains optimal performance with different sizes of training data. When the training data is small, the improvement achieved by MC-SQL over SQLova is more significant, especially on WikiSQL. In addition, the results on both datasets demonstrate that the less training data, the more significant the improvement brought by meta-learning. Note that changes in training data have less impact on ElectricitySQL than that on WikiSQL. It can be due to the few tables and the specific domain of ElectricitySQL.

Related Work

In recent research, mainstream text-to-SQL approaches mainly include two directions.

One direction is represented by Spider (Yu et al. 2018c), which is a benchmark to deal with the multi-table text-to-SQL task. This task aims to generate SQL queries according to the multiple tables joined by foreign keys. The generated queries typically include complex SQL syntaxes, such as GROUP BY and nested queries. Most existing multi-table text-to-SQL approaches (Yu et al. 2018a; Xu, Liu, and Song 2017; Yu et al. 2018b; Guo et al. 2019; Wang et al. 2020) follow the framework of Seq2Seq enhanced by grammar. However, even if the evaluation does not require to recognize the value in WHERE clause, the state-of-the-art performance (65.6% achieved by (Wang et al. 2020)) on this task is still far from realistic applications.

The other direction is represented by WikiSQL (Zhong, Xiong, and Socher 2017), which is a benchmark to deals with the single-table text-to-SQL task. This paper focuses on this task. Previous single-table text-to-SQL approaches (Zhong, Xiong, and Socher 2017; Xu, Liu, and Song 2017; Yu et al. 2018a; Dong and Lapata 2018; Chang et al. 2020) are mainly based on the Seq2Seq framework, trying to directly translate natural language questions into the tokens of the SQL queries. These approaches generate the entire SQL query by a single grammar-free Seq2Seq framework but ignore the characteristics of the SQL skeletons, thus their performance is limited. (Hwang et al. 2019) breaks the total task into several subtasks by taking advantage of the SQL skeleton of WikiSQL for the first time. They propose enables each sub-module to focus on the corresponding subtasks, thereby overcoming the bottleneck caused by a single model. In addition, the large-scale pre-trained language model (Devlin et al. 2019) also greatly improved model performance. Thereafter, almost all work on WikiSQL follows this framework of pre-trained models with multi-submodules. (He et al. 2019) leverages the type information of table headers and replaces BERT with MT-DNN, which is a stronger pre-trained model trained from multi-task learning. (Lyu et al. 2020) proposes a pair-wise ranking mechanism for each question-header pair, and achieve better results on WikiSQL. The significant difference between our work and these approaches is that we take advantage of the information of table content and improve the generalization ability of the model by meta-learning.

Conclusion

In this paper, we propose a new single-table text-to-SQL approach MC-SQL, which focuses on handling zero-shot tables. On the one hand, our approach takes advantage of table content to enhance the model. The potential header can be found by the semantic relevance of the question and table content. On the other hand, our approach learns the generalization capability from the different training tables by a meta-learning framework. The framework utilizes two-stage gradient update to force the model to generalize tables. The experimental results show that our approach achieved superior results than the baselines on both open-domain and domain-specific benchmark. More importantly, the improvements over the baselines further increase when dealing with zero-shot tables. In future work, we will try to classify different tables and combine meta-learning and reinforcement learning to further explore the generalization capabilities.

References

Chang, S.; Liu, P.; Tang, Y.; Huang, J.; He, X.; and Zhou, B. 2020. Zero-Shot Text-to-SQL Learning with Auxiliary Task. In *The 34th AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*, 7488–7495. AAAI Press. URL https://aaai.org/ojs/index.php/AAAI/article/view/6246.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J.; Doran, C.; and Solorio, T., eds., *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019*, 4171–4186. Association for Computational Linguistics. doi:10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Dong, L.; and Lapata, M. 2018. Coarse-to-Fine Decoding for Neural Semantic Parsing. In Gurevych, I.; and Miyao, Y., eds., *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018*, 731–742. Association for Computational Linguistics. doi:10.18653/v1/P18-1068. URL https://www.aclweb.org/anthology/P18-1068/.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, 1126–1135. PMLR. URL http://proceedings.mlr.press/v70/finn17a.html.

Guo, J.; Zhan, Z.; Gao, Y.; Xiao, Y.; Lou, J.; Liu, T.; and Zhang, D. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In Korhonen, A.; Traum, D. R.; and Màrquez, L., eds., *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28-August 2, 2019*, 4524–4535. Association for Computational Linguistics. doi:10.18653/v1/p19-1444. URL https://doi.org/10.18653/v1/p19-1444.

He, P.; Mao, Y.; Chakrabarti, K.; and Chen, W. 2019. X-SQL: reinforce schema representation with context. *CoRR* abs/1908.08113. URL http://arxiv.org/abs/1908.08113.

Hwang, W.; Yim, J.; Park, S.; and Seo, M. 2019. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization. *CoRR* abs/1902.01069. URL http://arxiv.org/abs/1902.01069.

Liu, X.; He, P.; Chen, W.; and Gao, J. 2019. Multi-Task Deep Neural Networks for Natural Language Understanding. In Korhonen, A.; Traum, D. R.; and Màrquez, L., eds., *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019*, 4487–4496. Association for Computational Linguistics. doi:10.18653/v1/p19-1441. URL https://doi.org/10.18653/v1/p19-1441.

Lyu, Q.; Chakrabarti, K.; Hathi, S.; Kundu, S.; Zhang, J.; and Chen, Z. 2020. Hybrid Ranking Network for Text-to-SQL. *CoRR* abs/2008.04759. URL https://arxiv.org/abs/2008.04759.

Snell, J.; Swersky, K.; and Zemel, R. S. 2017. Prototypical Networks for Few-shot Learning. In Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 4077–4087. URL http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Kavukcuoglu, K.; and Wierstra, D. 2016. Matching Networks for One Shot Learning. In Lee, D. D.; Sugiyama, M.; von Luxburg, U.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 3630–3638. URL http://papers.nips.cc/paper/6385-matching-networks-for-one-shot-learning.

Wang, B.; Shin, R.; Liu, X.; Polozov, O.; and Richardson, M. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In Jurafsky, D.; Chai, J.; Schluter, N.; and Tetreault, J. R., eds., *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 7567–7578. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.acl-main.677/.

Wang, C.; Huang, P.; Polozov, A.; Brockschmidt, M.; and Singh, R. 2018. Execution-Guided Neural Program Decoding. *CoRR* abs/1807.03100. URL http://arxiv.org/abs/1807.03100.

Xu, X.; Liu, C.; and Song, D. 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. *CoRR* abs/1711.04436. URL http://arxiv.org/abs/1711.04436.

Yu, T.; Li, Z.; Zhang, Z.; Zhang, R.; and Radev, D. R. 2018a. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation. In Walker, M. A.; Ji, H.; and Stent, A., eds., *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018*, 588–594. Association for Computational Linguistics. doi:10.18653/v1/n18-2093. URL https://doi.org/10.18653/v1/n18-2093.

Yu, T.; Yasunaga, M.; Yang, K.; Zhang, R.; Wang, D.; Li, Z.; and Radev, D. R. 2018b. SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-DomainText-to-SQL Task. *CoRR* abs/1810.05237. URL http://arxiv.org/abs/1810.05237.

Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; Zhang, Z.; and Radev, D. R. 2018c. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In Riloff, E.; Chiang, D.; Hockenmaier, J.; and Tsujii, J., eds., *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 3911–3921. Association for Computational Linguistics. doi:10.18653/v1/d18-1425. URL https://doi.org/10.18653/v1/d18-1425.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR* abs/1709.00103. URL http://arxiv.org/abs/1709.00103.