Zero-Shot Text-to-SQL Learning with Auxiliary Task

Shuaichen Chang,[1]* **Pengfei Liu,**[2] **Yun Tang,**[3] **Jing Huang,**[3] **Xiaodong He,**[3] **Bowen Zhou**[3]
[1]The Ohio State University, [2]Fudan University, [3]JD.COM AI Research
chang.1692@osu.edu, pfliu14@fudan.edu.cn, {yun.tang, jing.huang, xiaodong.he, bowen.zhou}@jd.com

Abstract

Recent years have seen great success in the use of neural seq2seq models on the text-to-SQL task. However, little work has paid attention to how these models generalize to realistic unseen data, which naturally raises a question: does this impressive performance signify a perfect generalization model, or are there still some limitations?

In this paper, we first diagnose the bottleneck of the text-to-SQL task by providing a new testbed, in which we observe that existing models present poor generalization ability on rarely-seen data. The above analysis encourages us to design a simple but effective auxiliary task, which serves as a supportive model as well as a regularization term to the generation task to increase the models′ generalization. Experimentally, We evaluate our models on a large text-to-SQL dataset WikiSQL. Compared to a strong baseline coarse-to-fine model, our models improve over the baseline by more than 3% absolute in accuracy on the whole dataset. More interestingly, on a zero-shot subset test of WikiSQL, our models achieve 5% absolute accuracy gain over the baseline, clearly demonstrating its superior generalizability.

Introduction

Text-to-SQL has recently attracted much attention as a sequence-to-sequence learning problem due to its practical usage for search and question answering (Dong and Lapata 2016; Zhong, Xiong, and Socher 2017; Xu, Liu, and Song 2017; Cai et al. 2018; Yu et al. 2018a; Dong and Lapata 2018; Finegan-Dollak et al. 2018; Yu et al. 2018b; Wang et al. 2017; Shi et al. 2018; McCann et al. 2018). The performance on some text-to-SQL tasks has been improved progressively (Dong and Lapata 2018; Wang et al. 2017; Shi et al. 2018; Hwang et al. 2019; He et al. 2019) in recent years. As pointed out in (Finegan-Dollak et al. 2018), when evaluating models on text-to-SQL tasks, we need to measure how well the models generalize to realistic unseen data, which is very common in the real applications.

Most of the previous text-to-SQL tasks assumed that all questions came from a fixed database and hence share one global table schema. This assumption is useful for some specific applications such as booking flights (Dahl et al. 1994) and searching GEO (Zelle and Mooney 1996), but not applicable to many real scenarios when different questions involve querying on different tables. (Zhong, Xiong, and Socher 2017) addressed this problem and generated a dataset called WikiSQL, which is by far the largest text-to-SQL benchmark dataset.

In WikiSQL many tables have different table schemas and each table has its own limited labeled data. One common approach is to encode the table column names in the input to the training of an encoder-decoder model (Yu et al. 2018a; Dong and Lapata 2018). (Yu et al. 2018a) proposed to utilize high-level type information to better understand rare entities and numbers in the natural language questions and encode these information from the input. These type information come from either external knowledge graph, a column or a number. This approach of TypeSQL (Yu et al. 2018a) was proven to be effective on WikiSQL when it is required for the model to generalize to new tables.

We observe that a text-to-SQL encoder-decoder model implicitly learn a mapping between entities in natural language questions to column names in tables. The model is likely to fail on mapping to new table column names that it never sees before. Hence if we learn a better mapping from question words to table column names, then the text-to-SQL generation model would be better generalized to new tables. With this in mind, we introduce an auxiliary model to enhance the existing generation model.

Specifically, we propose a novel auxiliary **mapping task** besides traditional text-to-SQL **generation task**. Here we explicitly model the mapping from natural language entities to table column names. The mapping model serves as an supportive model to the specific text-to-SQL task as well as regularization to the generation model to increase its generalization. These two tasks are trained together with a multi-task learning loss. In practice, we adopt the coarse-to-fine decoder as the prototype of our generation model due to their superior performance in text-to-SQL tasks. And the generation model is further improved by introducing bi-attention layer (question-to-table attention and table-to-question attention) (Seo et al. 2017) and attentive pooling layer (dos

*Work done during an internship at JD AI Research

SELECT $AGG $SEL
(**WHERE** $COND_COL $COND_OP $COND_VAL)
(**AND** $COND_COL $COND_OP $COND_VAL)*

Figure 1: SQL Sketch. The tokens starting with "$" are slots to fill. "*" indicates zero or more **AND** clauses.

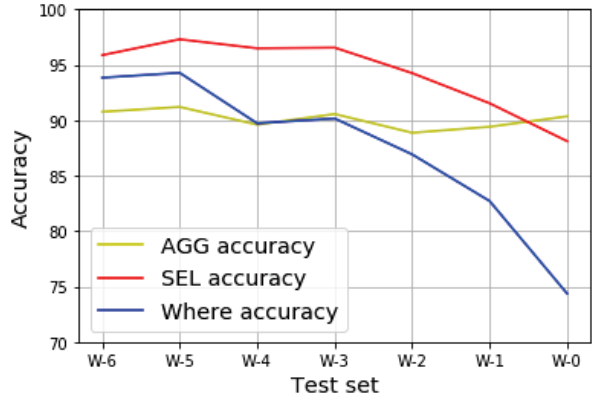


Figure 2: Break down accuracy of a strong baseline model (Dong and Lapata 2018). X-axis represents different subsets of WikiSQL test set, split by how many times a table occurs in training data. Splitting details are in Table 2.

Santos et al. 2016).

We test our models on WikiSQL, with emphasis on a ZERO-SHOT subset, where the table schemas of the test data never occur in the training data. Compared to the coarse-to-fine model, our models improve over the baselines by 3% absolute in accuracy, achieve execution accuracy of 81.7%. In particular, on the ZERO-SHOT test part of WikiSQL, our models achieve even more gain, with 5% improvement in accuracy over the baseline model. [1]

In summary our contributions in this paper are three-fold:

1) We find the existing testbed covers up the true generalization behavior of neural text-to-SQL models, and propose a new *zero-shot* test setting.

2) We improve the generalization ability of existing models by introducing an auxiliary task, which can explicitly learn the mapping between entities in the question and column names in the table.

3) The *zero-shot* evaluation not only shows the superior performance of our proposed method compared with the strong baseline but makes it possible to explain where the major gain comes from.

Background

Text-to-SQL Task

Text-to-SQL task can be formulated as a conditional text generation problem, in which a question $\mathcal{Q}$ and a table $\mathcal{C}$ are given, the goal is to generate a SQL language $\mathcal{Y}$.

[1] Our code can be found in https://github.com/JD-AI-Research-Silicon-Valley/auxiliary-task-for-text-to-sql

Figure 1 illustrates WikiSQL output format which consists of three components: AGG, SEL, and WHERE. Particularly, WHERE clause contains multiple conditions where each condition is a triplet with the form of (condition_column, condition_operation, condition_value).

Encoding Layer The question $\mathcal{Q}$ and corresponding table schema $\mathcal{C}$ are first translated into the hidden representation by a BiLSTM sentence encoder:

$$\mathbf{h}_t^q = \texttt{BiLSTM}(\overrightarrow{\mathbf{h}}_{t-1}^q, \overleftarrow{\mathbf{h}}_{t+1}^q, \mathbf{q}_t, \theta)$$
$$\mathbf{h}_t^C = \texttt{BiLSTM}(\overrightarrow{\mathbf{h}}_{t-1}^C, \overleftarrow{\mathbf{h}}_{t+1}^C \mathbf{C}_t, \theta)$$

where $\mathbf{q}_t$ is embedding of question word q_t and $\mathbf{C}_t$ is the representation of a column name C_t which consists of words $c_t^1, \cdots, c_t^{|C_t|}$. The first and last hidden state of a BiLSTM over C_t is concatenated as $\mathbf{C}_t$.

Decoding Layer Different from traditional text generation tasks, which share a decoder cross time-steps, in Text-to-SQL task, different decoders are designed in terms of different operations. Generally, these decoders can be classified two types: CLS for classifier, and PT for pointer.

CLS is used for the operations, such as AGG and COND_OP:

$$\text{CLS}(\mathbf{h}_t^d, \theta) = \texttt{softmax}(\mathbf{h}_t^d, \theta) \quad (1)$$

where $\mathbf{h}_t^d$ is one decoder hidden representation.

PT can be used to choose a proper column or word from a set of column or words. Formally, We refer to $\mathbf{h}_t^d$ as a pointer-query vector and $K = \{k_1, ..k_{|K|}\}$ as a set of pointer-key vectors, and predict the probability of choosing each key:

$$\text{PT}(\mathbf{h}_t^d, \mathbf{K}) = \texttt{softmax}(u) \quad (2)$$

u_i can be obtained as:

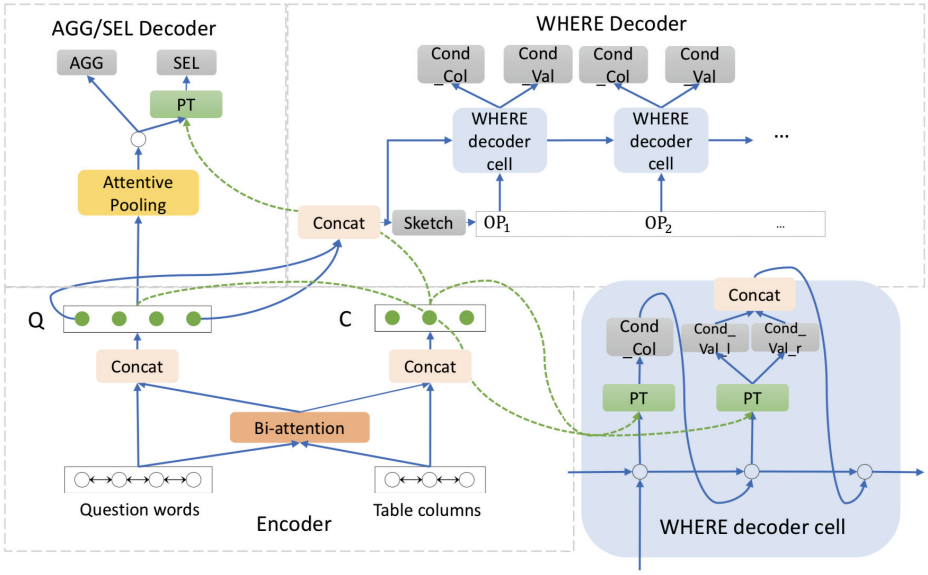
$$\mathbf{u}_i = \mathbf{v}^T \tanh\left(\mathbf{W}[\mathbf{h}_t^d, k_i] + b\right), \quad i \in (1, ..., |K|) \quad (3)$$

Diagnosing the Bottleneck of Text-to-SQL

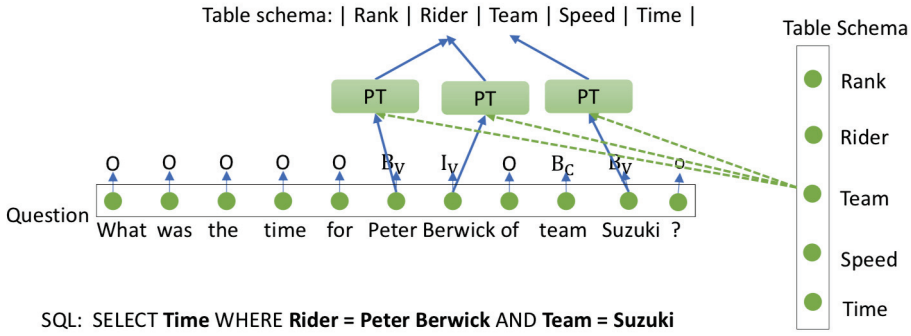
The existing testbed covers up the true generalization behavior of existing models. To address this problem, we provide a new testbed by breaking down the testing samples. Specifically, we analyze the generalization problem on table aware *Text-to-SQL* tasks, by testing previous state-of-the-art model (Dong and Lapata 2018) on different tables which occur different times in training set. We observe the following problems based on Figure 2:

- WHERE clause performance is more sensitive to how many times the table has been seen in the training data;
- The performance of WHERE would get a big drop once the table in test set is not present in the training data, i.e. zero-shot testing case.

Despite of the importance of the generalization problem of unseen tables, few work explored it due to the lack of appropriate datasets. The WikiSQL dataset was originally constructed to ensure that the training and test set have disjoint



(a)



(b)

Figure 3: Illustration of our model. The upper figure is the text-to-SQL generation model which consists of three parts: encoder (lower left), AGG/SEL decoder (upper left) and where decoder (upper right). Lower right is `WHERE` decoder cell. The bottom figure is our auxiliary mapping model with the ground-truth label of an example. Question word is mapped to a column only when it is tagged as part of a condition value (B_v or I_v).

set of tables, which can provide a test bed for generalization test of new tables. However, we find that the current version of WikiSQL test cannot guarantee this because different tables extracted from different wiki pages may share the same table schema (i.e. table column names), even though their table content may not be the same.

The above problems motivate us to explicitly model the mapping between words in question and table column names, and test the model generalization to new tables on the true zero-shot sub testset of WikiSQL.

Model

Our model consists of a seq2seq model for the SQL generation task (largely following the baseline coarse-to-fine model), and a mapping model as a auxiliary task to explicitly map question words to table schema (column names).

Main Generation Model

`Encoder` we follow the background section to obtain question and schema hidden representation $\mathcal{H}^q$ and $\mathcal{H}^c$. To enhance the interaction between question words q and column name c, a bi-attention is used to generate final question and table schema representation:

$$\bar{\mathcal{H}}^q, \bar{\mathcal{H}}^c = \texttt{BiAtt}(\mathcal{H}^q, \mathcal{H}^c, \theta)$$

Considering the nature of structured SQL, we follow previous works to use different sub-decoders for `AGG`, `SEL` and `WHERE` clause. Especially, our `WHERE` decoder is adapted from the baseline model (Dong and Lapata 2018).

`AGG` and `SEL` Decoder Each SQL only contains one `AGG` and `SEL`, so we generate `AGG` and `SEL` based on entire question representation. Since different words or phrases in question do not equally contribute to the decisions of `AGG` and

SEL, we employ an attentive pooling layer over $\bar{\mathcal{H}}^q$ to generate final hidden representation $\mathbf{q}^{SEL}$ for AGG and SEL.

We feed $\mathbf{q}^{SEL}$ into CLS layer to generate the aggregation operation AGG and meassure the similarity score between $\mathbf{q}^{SEL}$ and each column name $\bar{\mathbf{C}}_j$ to predict SEL with PT layer in (2):

$$y^{AGG} = \text{CLS}(\mathbf{q}^{SEL}, \theta)$$
$$y^{SEL} = \text{PT}(\mathbf{q}^{SEL}, \bar{\mathcal{H}}^c)$$

WHERE Decoder We take the WHERE decoder from the-state-of-the-art model (Dong and Lapata 2018), which first generates a slot sketch of WHERE clause and transform the SQL generation into a slot filling problem. There are 35 categories of WHERE clauses in WikiSQL and each one is subsequence of WHERE clause which skips the COND_COL and COND_VAL. For example, "WHERE = AND >" is a sketch of WHERE clause which has 2 conditions. We first predict the sketch α based on $\bar{\mathcal{H}}^q$:

$$y^{\alpha} = \text{CLS}(\mathbf{q}^{Where}, \theta),$$

where $\mathbf{q}^{Where} = [\bar{\mathbf{h}}^q_1, \bar{\mathbf{h}}^q_{|\mathcal{Q}|}]$.

Once y^{α} is predicted, we obtain the COND_OP sequence it represents. We embed each operation in COND_OP sequence and feed them into WHERE-decoder cell. As Figure 3 shows, the WHERE-decoder cell takes one COND_OP as input and output COND_COL and COND_VAL for each decoder time step, while each decoder time step has 3 LSTM time steps. For the ith condition, $x^d_{i,1}$,$x^d_{i,2}$,$x^d_{i,3}$ are COND_OP_i and COND_COL_i and COND_VAL_i and output $y^d_{i,1}$,$y^d_{i,2}$ are probability distribution of the index of COND_COL_i and the span of COND_VAL_i. We do not have output for each $y^d_{i,3}$ because the input of next time step is given by pre-predicted COND_OP_{i+1}. The lstm-cell is updated 3 times inside the WHERE-decoder cell for each decoder time step:

$$h^d_{i,j} = \begin{cases} \text{LSTM}(x^d_{i,j}, h^d_{i,j-1}) & j \neq 1 \\ \text{LSTM}(x^d_{i,j}, h^d_{i-1,3}) & j = 1 \end{cases}$$

The output layers for COND_COL and COND_VAL are both pointer layer which are pointed to column names and question words to predict COND_COL index and the left and right end VAL^l, VAL^r of the span of COND_VAL in question:

$$y^d_{i,1} = \text{PT}(h^d_{i,1}, \mathcal{H}^c)$$
$$y^d_{i,2} = P(VAL^l_i|\cdot) \cdot P(VAL^r_i|VAL^l_i, \cdot)$$
$$P(VAL^l_i|\cdot) = \text{PT}(h_{i,2}, \mathcal{H}^q)$$
$$P(VAL^r_i|VAL^l_i, \cdot) = \text{PT}([h_{i,2}; \bar{\mathbf{h}}^q_{VAL^l_i}], \mathcal{H}^q)$$

Auxiliary Mapping Model

For a SQL query, each condition consists of three parts, COND_COL, COND_OP and COND_VAL. Our mapping model aims to discover the mapping between condition column and condition value. The mapping prediction is based on question and table schema representation $\mathcal{H}^q$ and $\mathcal{H}^c$, which are shared with generation model. Optimization based on mapping task loss can improve the question and table schema representation. An intuitive way to achieve mapping is to directly learn a mapping function from each word in question to column names. However, since not all words in a question are condition values, it's challenging to take all words into consideration. To address this problem, we propose a two-step mapping model, in which we first learn a detector to screen out condition values, and then we learn a mapping function from condition values to column names.

Condition Value Detection Because the condition value sometimes covers multiple words, we label the span for condition values in questions with typical BIO (Nadeau and Sekine 2007) tags. We notice sometimes condition column names appear exactly in question, so the span of column name in question is also labeled with tags B_c, I_c during training when a column name appear in question. Altogether we have five tags B_c, I_c, B_v, I_v, O, which represent the first word of condition column, subsequent word of condition column, the first word of condition value, subsequent word of condition value and outside, respectively. Figure 3 illustrates our mapping model by giving the ground-truth label for an example.

The mapping model takes encoding vector of question words $\bar{\mathcal{H}}^q = \bar{\mathbf{h}}^q_1, ..., \bar{\mathbf{h}}^q_{|\mathcal{Q}|}$ and column names $\bar{\mathcal{H}}^c = \bar{\mathbf{h}}^c_1, ..., \bar{\mathbf{h}}^c_{|\mathcal{C}|}$ as input. Mapping model first predict gate y^{tag}:

$$y^{tag}_i = \texttt{argmax}(\mathbf{v}_{tag} \tanh(\mathbf{W}_{tag} \bar{\mathbf{h}}^q_i + \mathbf{b}_{tag})),$$

where $\mathbf{W}_{tag} \in \mathbb{R}^{5*H}$ and $\mathbf{b}_{tag} \in \mathbb{R}^5$ are tagging parameters.

Value-column Mapping We only learn the mapping for question words which are tagged as B_v, I_v:

$$y^{map}_i = \text{PT}(\bar{\mathbf{h}}^q_i, \bar{\mathcal{H}}^c), \quad y^{tag}_i \in \{B_v, I_v\}$$

Loss Function

We refer to the following L_{gen} as generation task loss and $\mathcal{L}_{map}$ as mapping task loss.

$$L_{gen} = -\sum_{i=1}^{|\mathcal{Y}|} y^{op}_i \log(\hat{y}^{op}_i),$$
$$L_{map} = -\sum_{i=1}^{|\mathcal{Q}|} y^{tag}_i \log(\hat{y}^{tag}_i) - \sum_{i=1}^{K} y^{map}_i \log(\hat{y}^{map}_i),$$

where op represents different operations during decoder phase. y and $\hat{y}$ denote the probability distribution of real label and predicted probability distribution. K represents how many times words in question have been predicted as condition values.

Finally, the overall loss can be written as:

$$\mathcal{L} = \sum_{i=1}^{N} \lambda L_{gen} + (1 - \lambda) L_{map}$$

where N is the number of training samples and λ is hyper-parameter.

Experimental Setup

Dataset

WikiSQL has over 20K tables and 80K questions corresponding to these tables. This dataset was designed for translating natural language questions to SQL queries using the corresponding table columns without access to the table content. This dataset is further split into training and testing sets that are separately obtained from different Wiki pages, assuming there is no overlap of tables between training and testing sets. However, we find in this split, 70% question-table pairs in test set have the same table schema as those in the training set. This is because even train and test tables were obtained from different Wiki pages, these tables could still have the same table schema. For example, different football teams have their own Wiki page but each one have a table with the same schema recording match information.

We split the test set based on the number of shots (the number of a table occurrences in training data). We report experiments on the original full WikiSQL test set as well as different subset based on the number of shots, especially on the *zero-shot* testing case. Statistics of new test sets are in table 2.

Evaluation

We follow the evaluation metrics in (Xu, Liu, and Song 2017) to measure the query synthesis accuracy: query-match accuracy ($\mathrm{ACC}_{\mathrm{qm}}$) which measures the decoded query match the ground truth query without considering the order of conditions and execution accuracy ($\mathrm{ACC}_{\mathrm{ex}}$) which measures the results from executing predicted queries. The accuracies are further broken down into three categories: `AGG`, `SEL` and `WHERE`, as in (Xu, Liu, and Song 2017).

Model Configuration

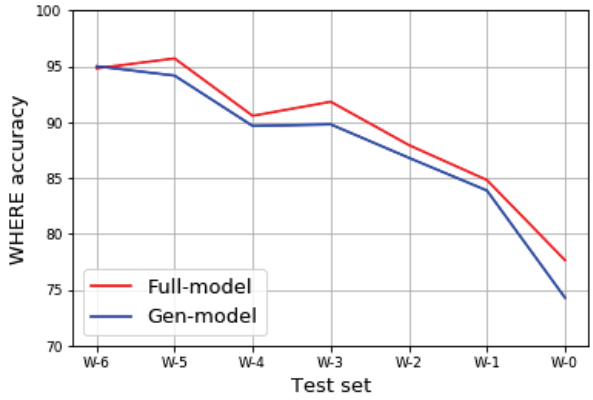
We use 300-dim Glove word embedding as our pre-trained embedding. Hidden size for all LSTM is 250 and hidden size in attention function is set to 64. The loss weight λ is set to 0.5. A 0.5-rate dropout layer is used before each output layer. Each concatenation is followed by one full-connected layer to reduce the dimension to the original hidden or attention size. Test model is selected by the best performing model on validation set.

Overall Results and Analysis

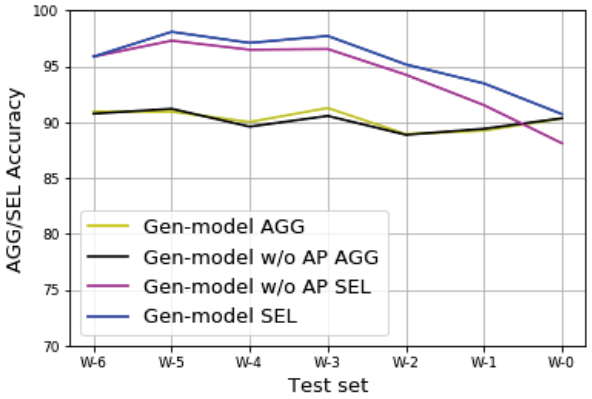
Table 1 shows the overall and breakdown results on full WikiSQL dataset. We compare our models with strong baseline models on the original WikiSQL test data. All these models have no access to table content following (Zhong, Xiong, and Socher 2017).

First our Gen-model with enhanced encoder/decoder improves over the baseline coarse-to-fine model by 1.6% in accuracy of both ACC_{qm} and ACC_{ex}. Our Gen-model mainly improves on ACC_{SEL} compared to baseline models. Ablation test shows it is attributed to the attentive pooling in `SEL` decoding.

Second our Full-model outperforms our single generation model by 1.5% and 1.6% in query-match accuracy and execution accuracy, achieving a very competitive new execution accuracy of 81.7%. Break down results show Full-model mainly improves the accuracy over Gen-model on the `WHERE` clause, with 1.9% accuracy gain. Break down results illustrate the function of mapping task, which is to connect condition value with condition column explicitly.



(a) WHERE DECODER



(b) AGG/SEL DECODER

Figure 4: Accuracy of Full-model and Gen-model in different test subsets. W-0 represents zero-shot setting. The frequency of the table has been seen in the training data decrease from W-6 to W-0.

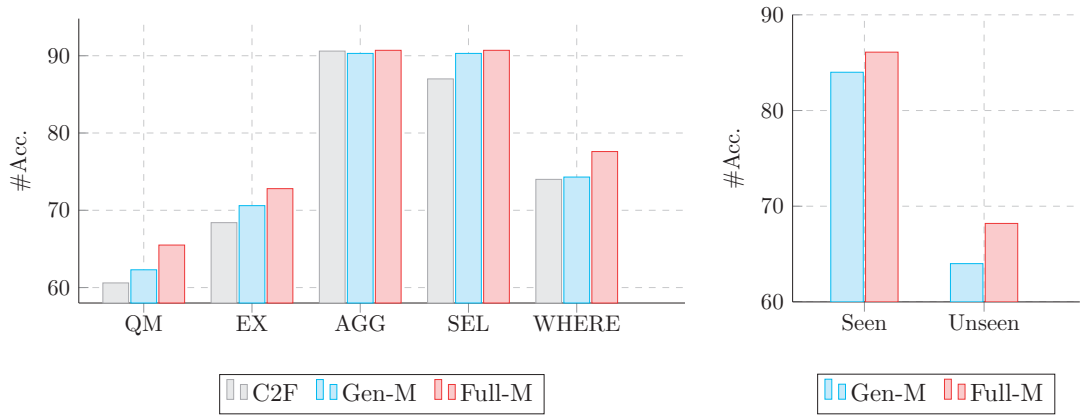
Training data amount

Figure 4a illustrates Gen-model and Full-model accuracy of `WHERE` clause prediction on different test subsets from Table 2. Full-model is consistently better than single Gen-model in `WHERE` clause prediction. The biggest gap between Full-model and Gen-model in `WHERE` clause accuracy is on test subset W-0. This shows that Full-model generalizes better than Gen-model for the unseen test tables. We also found that Full-model accuracy on W-4 is slightly lower than that on W-3. We believe this is due to the fact that table itself is the other fact affecting models' performance, in addition to the amount of training tables.

Figure 4b again illustrates Gen-model outperforms Gen-model without attentive pooling on different amount of

Model	ACC_{qm}	ACC_{ex}	ACC_{agg}	ACC_{sel}	ACC_{where}
SEQ2SQL (Zhong, Xiong, and Socher 2017)	-	59.4%	90.1%	88.9%	60.2%
SQLNET (Xu, Liu, and Song 2017)	61.3%	78.0%	90.3%	90.9%	71.9%
TypeSQL (Yu et al. 2018a)	66.7%	73.5%	90.5%	92.2%	77.8%
COARSE2FINE (Dong and Lapata 2018)	71.7%	78.5%	90.4%	92.4%	84.2%
Gen-model w/o AP	72.8%	79.4%	90.2%	93.0%	84.7%
Gen-model	73.5%	80.1%	90.3%	94.2%	84.8%
Full-model	**75.0%**	**81.7%**	90.5%	**94.5%**	**86.7%**

Table 1: Overall and break down results on full WikiSQL dataset. ACC_{qm}, ACC_{ex} are accuracy numbers of query match (ignore the order of conditions) and execution result, and ACC_{agg}, ACC_{sel}, ACC_{where} are the accuracy of AGG, SEL, WHERE clauses. The upper part are baseline models, and the lower part are our generation model Gen-model and the whole model Full-model which is the Gen-model with the auxiliary mapping model. Gen-model w/o AP is the generation model without attentive pooling.



(a) Results on Unseen tables (W-0)

(b) Results on seen/unseen columns

Figure 5: *C2F*, *Gen-M* and *Full-M* represent the baseline coarse-to-fine model, and our proposed Gen-model and Full-model respectively.

training data.

Zero-shot Test

After analyzing the function of our attentive pooling and auxiliary mapping model, we want to know the effectiveness of the mapping model in zero-shot setting.

Figure 5a illustrates the results on *zero-shot* test case (i.e. W-0). Our Full-model outperforms baseline coarse-to-fine model by 4.9% and 4.4% in ACC_{qm} and ACC_{ex}. The accuracy improvement over the baseline coarse-to-fine model lie on the SEL and WHERE clause. Full-model achieves 3.6% WHERE accuracy gain for unseen data over the Gen-model while it outperforms Gen-model by 1.1% on WHERE clause for seen data (1.9% improvement on all data), which shows the generalization ability of the auxiliary mapping model on zero-shot setting.

Figure 4a shows WHERE clause accuracy has a big drop on zero-shot setting (W-0) compared to few-shot setting (W-1). We further analyze the reason of this degradation by looking into how the performance is affected by whether a column name is present in the training data. On unseen test table schema, 28% column names never appear in training set, which makes question related to these columns harder. We further divide conditions in WHERE clauses into two classes, one class with condition column appearing in training, the other with condition column not appearing in training. We measure the accuracy of each class on the WHERE clause. The result is reported in Figure 5b. Full-model outperforms single generation model by 4.2% on unseen column names and 2.1% on seen column names. Full-model even shows the better generalization ability for harder data (column names never appear in training set) comparing to seen columns

dataset	number of shots	#questions
W-full	[0,2045]	15878
W-0	0	5201
W-1	[1,5]	1700
W-2	[6,15]	1842
W-3	[16,40]	1971
W-4	[41,100]	1654
W-5	[101,500]	1887
W-6	[501,2045]	1623

Table 2: Statisitics of WikiSQL test set. W-full is original WikiSQL test set and W-0, W-1,$\cdots$, W-6 are subsets split by the number of shots (number of a table occurrences in the training data).

Examples	(a)	(b)	(c)	others
Case-Wrong	63	22	18	4
Case-Correct	71	19	10	3

Table 3: Number of samples in each error categories.

names.

Case Study on Zero-shot Setting

We also manually analyze the Full-model behavior on zero-shot test compared to the Gen-model alone. We first randomly sample 100 examples of which Full-model predicts correct on `WHERE` clause (Case-Correct in Table 3), while Gen-model fails. We label the failure reasons of Gen-model into four categories (one example can belong to more than one categories): (a) wrong `COND_COL` prediction, (b) wrong `COND_VAL` prediction, (c) predicting extra conditions or missing conditions and (d) others. Table 3 shows the majority of `WHERE` clause errors are in (a): wrong COND column name errors. We then randomly sample another set of 100 examples (Case-Wrong in Table 3): Gen-model predicts `WHERE` clause correctly on these examples but Full-model fails. Table 3 indicates Full-model corrects Gen-model mainly on wrong `COND_COL` prediction, which shows our mapping task improves column name predictionin the generation task.

Related Work

Recently neural network based approaches, especially seq-to-seq models have been applied to text-to-SQL successfully (Wang, Cheung, and Bodik 2017; Neelakantan et al. 2017; Iyer et al. 2017; Yin and Neubig 2017; Huang et al. 2018; Zhong, Xiong, and Socher 2017; Xu, Liu, and Song 2017; Cai et al. 2018; Yu et al. 2018a; Dong and Lapata 2018; Finegan-Dollak et al. 2018; Hwang et al. 2019).

Sketch-based approach is very effective, especially on WikiSQL task (Zhong, Xiong, and Socher 2017; Xu, Liu, and Song 2017; Yu et al. 2018a). In (Zhong, Xiong, and Socher 2017) SEQ2SQL model used a coarse-grained sketch: aggregation, SELECT column and `WHERE` clause; (Xu, Liu, and Song 2017) used a finer sketch to align to the syntactical structure of a SQL query with three specific slot-filling models: model_COL, model_AGG, and model_VAL. In TypeSQL (Yu et al. 2018a) it also adopted this sketch-based model structures. However, in (Dong and Lapata 2018) sketch was referred to as abstractions for meaning representation, leaving out low-level details. This meaning sketch was used as an input to the final decoding.

One challenge of using neural seq2seq models is the need of large annotated question-query pairs. (Zhong, Xiong, and Socher 2017; Cai et al. 2018) have automatically generated large datasets using templates and had humans paraphrased the questions into natural language questions. WikiSQL is by far the largest text-to-SQL dataset. WikiSQL was designed for testing model's generalization by splitting the tables in a way that there is no overlap of tables in training and testing.

Execution guided (EG) decoding was recently proposed in (Wang et al. 2017) that detects and excludes faulty outputs during the decoding by conditioning on the execution of partially generated output. Adding execution guided decoding to the coarse-to-fine model improved accuracy by 5.4% on the wikiSQL dataset; and adding on top of the most recent IncSQL model (Shi et al. 2018) improved accuracy by 3.4%. It is proven that the EG module is very effective with any generative model.

Zero-shot semantic parsing has not obtained enough attention. (Herzig and Berant 2018) applied a pipeline framework, including four independent models to achieve generalization, while our work is end-to-end trained and focusing on improving model's generalization with an auxiliary mapping task. Zero-shot slot filling (Bapna et al. 2017) also leverages the text of schema to connect language question words to column names (slots), but their model needs to predict the probability of each possible column indepentently while our model can select the column by processing the question and schema one time.

Conclusions and Future Work

In this paper, we propose a novel auxiliary mapping task for zero-shot text-to-SQL learning. Traditional seq2seq generation model is augmented with an explicit mapping model from question words to table schema. The generation model is first improved by an attentive pooling inside the question, and bi-directional attention flow to improve the interaction between the question and table schema. The mapping model serves as an enhancement model to text-to-SQL task as well as regularization to the generation model to increase its generalization.

We compare our models with the a strong baseline coarse-to-fine model on the original WikiSQL testset as well as on the totally unseen test tables (a subset of zero-shot testing). Experimental results show that our models outperform baseline models on both setting. Even though the generation model is already augmented with bi-directional attention to

enhance the interaction between question and table, our results and analysis demonstrate that the explicitly mapping task can further increase the capability of generalization to unseen tables.

Spider (Yu et al. 2018b) was recently proposed as another large cross-domain text-to-SQL dataset besides WikiSQL. It has more complex SQL templates including joint tables, which brings other interesting problems except for generalization. We plan to expand our models on this new dataset in the future.

References

Bapna, A.; Tur, G.; Hakkani-Tur, D.; and Heck, L. 2017. Towards zero-shot frame semantic parsing for domain scaling. *arXiv preprint arXiv:1707.02363*.

Cai, R.; Xu, B.; Yang, X.; Zhang, Z.; and Li, Z. 2018. An encoder-decoder framework translating natural language to database queries. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*.

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

Dong, L., and Lapata, M. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 33–43.

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

dos Santos, C. N.; Tan, M.; Xiang, B.; and Zhou, B. 2016. Attentive pooling networks. *CoRR*.

Finegan-Dollak, C.; Kummerfeld, J. K.; Zhang, L.; Ramanathan, K.; Sadasivam, S.; Zhang, R.; and Radev, D. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 351–360.

He, P.; Mao, Y.; Chakrabarti, K.; and Chen, W. 2019. X-sql: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*.

Herzig, J., and Berant, J. 2018. Decoupling structure and lexicon for zero-shot semantic parsing. *arXiv preprint arXiv:1804.07918*.

Huang, P.-S.; Wang, C.; Singh, R.; Yih, W.-t.; and He, X. 2018. Natural language to structured query generation via meta-learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 732–738.

Hwang, W.; Yim, J.; Park, S.; and Seo, M. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Iyer, S.; Konstas, I.; Cheung, A.; Krishnamurthy, J.; and Zettlemoyer, L. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 963–973.

McCann, B.; Keskarm, N. S.; Xiong, C.; and Socher, R. 2018. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv::1806.08730v1*.

Nadeau, D., and Sekine, S. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes* 30(1):3–26.

Neelakantan, A.; Le, Q. V.; Abadi, M.; MacCallum, A.; and Amodei, D. 2017. Learning a natural language interface with neural programmer. In *Proceedings of the 5th International Conference on Learning Representations*, 1–10.

Seo, M.; Kembhavi, A.; Farhadi, A.; and Hajishirzi, H. 2017. Bidirectional attention flow for machine comprehension. In *Proceedings of the 5th International Conference on Learning Representations*.

Shi, T.; Tatwawadi, K.; Chakrabarti, K.; Mao, Y.; and Polozov, Oleksandr, a. C. W. 2018. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *arXiv preprint arXiv:1809.05054v2*.

Wang, C.; Kedar, T.; Marc, B.; Po-Sen, H.; Yi, M.; Oleksandr, P.; and Rishabh, S. 2017. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100v3*.

Wang, C.; Cheung, A.; and Bodik, R. 2017. Synthesizing highly expressive sql queries from input-output examples. In *ACM SIGPLAN Notices*, volume 52, 452–466. ACM.

Xu, X.; Liu, C.; and Song, D. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Yin, P., and Neubig, G. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 440–450.

Yu, T.; Li, Z.; Zhang, Z.; Zhang, R.; and Radev, D. 2018a. Typesql: knowledge-based type-aware neural text-to-sql generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 588–594.

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. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3911–3921.

Zelle, J. M., and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, 1050–1055.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.