ST-SQL: Semi-Supervised Self-Training for Text-to-SQL via Column Specificity Meta-Learning

Anonymous ACL submission

Abstract

The few-shot problem is an urgent challenge for the generalization capability of the single-table text-to-SQL task. Current few-shot methods neglect the potential information of unlabeled data and have a domain bias due to the same weight of samples. Motivated by this, this paper proposes a Self-Training text-to-SQL (ST-SQL) method which handles the problem from both views of data and algorithms. At the data level, ST-SQL performs data expansion by using an iterative framework to attach pseudo-labels to unlabeled data. The expanded data are sampled to reversely train the model. At the algorithm level, ST-SQL defines a column specificity to perform a more fine-grained gradient update during meta-training. The common samples are attached more weight to eliminate the domain bias. ST-SQL achieves state-of-the-art results on both open-domain and domain-specific benchmarks and brings more significant improvements on few-shot tests.

1 Introduction

Single-table text-to-SQL is a widely-studied semantic parsing task, which aims to translate natural language questions (NLQ) into SQL programs to access a single table. WikiSQL (Zhong et al., 2017) is the currently largest single-table text-to-SQL dataset, each example of which consists of a table, a question, and a target SQL program to be obtained. Figure 1 shows an example in WikiSQL.

Recently, benefiting from tabular-specific pre-trained models (Yin et al., 2020; Herzig et al., 2020; Zhao et al., 2021; Yu et al., 2020) and well-designed strategies (Hui et al., 2021; Chen et al., 2021a; Deng et al., 2021), state-of-the-art methods have been able to achieve more than 90% SQL execution accuracy on full-set WikiSQL. It brings a view that the single-table task seems to be solved (Suhr et al., 2020), so more and more works begin to challenge harder multi-table tasks (Yu et al., 2018b; Shi et al., 2021; Chen et al., 2021b).

However, is this really the case? We observe that most existing text-to-SQL models depend on sufficient pairs of NLQ and SQL annotations, which are regarded as the supervision training signal. Unfortunately, for each table in the actual scenario, high-quality SQL annotations require a high manual cost, consequently, there are only a few NLQ-SQL pairs that can be used for training. Some works (Chang et al., 2020; Chen et al., 2021a) call it a few-shot table challenge and demonstrate that it directly hinders the single-table text-to-SQL technology from theory to application.

On the one hand, although pre-trained models designed for tabular data (Yin et al., 2020; Herzig et al., 2020; Zhao et al., 2021; Yu et al., 2020) can bring a wealth of prior knowledge, they require millions of parallel corpus of text and tables for pre-training. It is obviously difficult to implement in some specific domains or non-English scenes. On the other hand, Meta-Learning (ML) is also a natural way to handle few-shot tables. Regrettably, the current ML-based framework (Chen et al., 2021a) is one-size-fits-all for all sub-tasks of text-to-SQL while ignoring their potential priorities. More importantly, a fact is neglected that different column samples typically have different domain relevance. For example in Figure 2, compared to column header “Market Value”, which is almost only found in financial tables, “Company” is more...
Figure 2: Column “Company” with low specificity can appear in tables with different topics, while column “Market Value” with high specificity may only exist in the financial tables.

common for all the tables. Intuitively, in order to make the model general to fast adapt to few-shot tables, the training process should pay more attention to the common columns, which are at the center of the sample space.

In this paper, we propose a new method called Self-Training text-to-SQL (ST-SQL) that overcomes the limitations by integrating the views of data and algorithms. We select Hydranet (Lyu et al., 2020), which is a strong text-to-SQL baseline, as our basic model and follow existing work to wrap it with content enhancement (Chen et al., 2021a).

In order to improve its few-shot performance from the view of data, we propose a semi-supervised learning framework called ST-Semi to promote it by utilizing the potential information of unlabeled data. Specifically, in each training iteration, the basic model first predicts the SQL program of unlabeled data. Then, it makes the model general to fast adapt to few-shot tables.

Some works (Lyu et al., 2020; Yin et al., 2020) have shown that SC and WC, which are directly related to columns, have the most important role in the final performance.

In this paper, the training data is denoted by \( D = (A, U) \), where \( A = \{a_1, a_2, ..., a_{|A|}\} \) is the set of labeled data and \( U = \{u_1, u_2, ..., u_{|U|}\} \) is the set of unlabeled data. Here, \( a_i = \{(q_i, y_i, T_i)\} \) and \( u_j = \{(q_j, T_j)\} \). \( y_i \) is the gold SQL for \( q_i \). Generally, \(|A| < |U|\).

For \( M \) with parameter...
where \( \theta \) is a set of parameters of \( \mathcal{F} \) used to obtain a new set of parameters \( \hat{\theta} \) as follows.
\[
\mathcal{M}'(\theta') = \mathcal{F}(\mathcal{M}(\theta), A, \mathcal{U})
\]

where \( \hat{\theta} \) has few-shot adaptation capability. Next, we will detail the basic model \( \mathcal{M} \) and the training framework \( \mathcal{F} \).

### 3.1 Basic Model

As mentioned in Section 2, the column prediction is critical for the total task. Therefore, we adopt Hydranet (Lyu et al., 2020) as our basic model \( \mathcal{M} \). In addition, in view of the proven improvement of the table content to the SQL task (Yu et al., 2020; Chen et al., 2021a), we apply a content enhancement (Chen et al., 2021a) for \( \mathcal{M} \). Specifically, the basic model consists of an encoding module and sub-tasks module.

#### 3.1.1 Encoding Module

The encoding module is set to a pre-trained RoBERTa. For the purpose of use the semantic information of each NLQ-column pair \((q, h_i)\), the input format is represented by:
\[
[CLS], x_{i1}, [BLK], x_{i2}, \ldots, x_{in}, [BLK], x_{i1}^d, \ldots,
\]
where \([CLS], [BLK] \) and \([SEP] \) are user-defined placeholders, \( x_{it} \) denotes the type of \( h_i \) and \( x_{iz}^d \) denotes the \( i \)-th token of \( z \). \( c_i^z \) denotes one cell of the column contents under \( h_i \), used to enhance the potential semantic information. In this paper, top-k cells are obtained by calculating a literal score (Chen et al., 2021a) with \( q \). The inputs are converted by RoBERTa into the following vectors:
\[
[h_{1/CLS}], h_{t1}, h_{BLK}, h_{t2}, \ldots, h_{in}, h_{BLK}, h_{t1}^d, \ldots,
\]
where each \( h_i \in \mathbb{R}^d \) is the hidden state of the input token \( x \) with the context information, and \( h_{1/CLS} \) is regarded as the semantic vector of \((q, h_i)\).

#### 3.1.2 Sub-Tasks Module

This module is used to make predictions for the sub-tasks in Section 2. First, for the SC task and WC task, the ranking scores of each column \( \mathcal{H}_i \) are calculated by
\[
P_{sc}(\mathcal{H}_i | q, h) = \text{sigmoid}(W_{sc} \cdot h_{1/CLS}^i),
\]
where \( W_{sc} \in \mathbb{R}^d \) and \( W_{wc} \in \mathbb{R}^d \) are trainable parameter matrices. These two scores are the basis of the whole SQL generation procedure. Then, the result of SC can be obtained by selecting the column with the highest \( P_{sc} \) score. Before getting the result of WC, WN needs to be calculated the first which can be calculated by a weighted sum of the conditional probability of each candidate number as follows.
\[
P_{wn}(n_j | q, h^*) = \text{softmax}(W_{wn} | j : h_{1/CLS}^i),
\]
where \( W_{wn} \in \mathbb{R}^{n \times d} \), \( w \in \mathbb{R}^d \) are affine transformations. \( P(n_j | q, h_i) \) denotes the probability of \( n_j \) that \( i \)-th column predicts, \( n \) denotes the maximum of candidate \( n \), and \( n' \) is the result for WN. Subsequently, the top \( n \) columns with the highest \( P_{wn} \) scores are taken as the result of WC. Finally, the remaining tasks can be predicted based on the columns selected by SC and WC as follows.
\[
P_{sa}(a_j | q, h^*) = \text{softmax}(W_{sa} | j : h_{1/CLS}^i),
\]
where \( W_{sa} \) are affine transformations. \( P_{sa} \) denotes the probability of \( h^* \) that \( i \)-th column predicts, and \( P_{na} \) is the result for SA. Both the results from WC and SA are then combined by a weighted sum to get the final result.

**Figure 3:** An overview of ST-SQL. a) The architecture of the basic model. b) The training procedure of ST-Semi.
where $W_{sa} \in \mathbb{R}^{n_a \times d}$, $W_{wo} \in \mathbb{R}^{n_o \times d}$, $W^s_{wo} \in \mathbb{R}^{d}$ and $W^e_{wo} \in \mathbb{R}^{d}$ are weight matrices, $n_a$ and $n_o$ denote the number of candidate aggregation functions and operators, $h^q_i \in \mathbb{R}^{b_q \times d}$ represent the question embedding for $i$-th column, $l_q$ denotes the length of the question. The aggregation function and operators with the highest $P^q_{sa}$ and $P^q_{wo}$ scores calculated according to the columns of SC and WC, are chosen as the result of AGG and OP respectively. Mentions for WV are extracted by the start and end indexes with the highest $P^s_{wo}$ and $P^e_{wo}$ scores. Finally, a SQL program is obtained by filling the results of the six sub-tasks into the SQL skeleton.

### 3.2 Self-Training Framework

Our proposed self-training framework ST-Semi, i.e., $\mathcal{F}$ in (1), integrates the meta-learning technique into a semi-supervised training process. Algorithm 1 shows the detailed procedure.

The entire process can be divided into a warm boot stage and an iterative adaptation stage.

In the warm boot stage (line 2-9), random initialized $\mathcal{M}$ is trained by a conventional mini-batch strategy on labeled data $\mathcal{A}$ (line 5-6) while its best performance $s$ on the validation set is updated (line 8). When $s$ reaches threshold $\lambda$, the procedure is completed and parameters $\theta_w$ are obtained. Here, we use LF accuracy (detailed in 4.1.2), which is a widely-used metric for text-to-SQL, to evaluate $s$.

In the iterative adaptation stage, $\mathcal{M}$ is trained by using both $\mathcal{A}$ and unlabeled data $\mathcal{U}$. Concretely, at the first step of each epoch, $\mathcal{M}$ predicts SQL program $\hat{y}$ as a pseudo label for each unlabeled sample $(q, T)$ in $\mathcal{U}$ (Line 12-17). The pseudo labeled data is used to reverse promote the parameter update of $\mathcal{M}$. Intuitively, pseudo-label samples with higher model prediction confidence can have a greater impact on parameter updating. Therefore, we design a confidence score for each sample as follows.

$$\psi_i = \frac{1}{\zeta} \prod_{z_i} \max(P(x_i | q_i, T_i), \zeta), \quad (11)$$

where $\zeta$ is a threshold and $\tau$ is a hyper-parameter. $P(x_i | q_i, T_i)$ is the confidence in the current task according to the model’s output, where $z_i \in \{\text{SC, WC, SA}\}$. Note that we only calculate $\psi_i$ for SC, WC, and SA because the first two tasks are very basic and important, and SA generally has a poor performance.

At the second step, $\theta_w$ is updated by the mixed data of the labeled data and pseudo labeled data. According to the characteristics of different sub-tasks, we update $\theta_w$ in two ways. For SC and WC, tasks $D_{\mathcal{P}}$ are first sampled from the mixed data $\mathcal{D}_{\mathcal{M}} = \mathcal{A} \cup \mathcal{P}$. $\theta_m$ is obtained by updating $\theta_w$ with column specificity meta-learning $\mathcal{F}_m$ on $\mathcal{D}_{\mathcal{P}}$ (Line 19-20). Furthermore, in order to stabilize the parameters and adapt $\mathcal{M}$ into the target dataset, for all sub-tasks, a mini-batch is performed after the meta-learning. Here, $\mathcal{M}$ is trained by $\mathcal{D}_m = \mathcal{A} \cup \mathcal{P'}$ in mini-batch form, where $\mathcal{P}'$ is a subset sampled from $\mathcal{P}$ (Line 21-25).

After multiple epochs of iterative adaptation, $\theta'$, which is the set of parameters with the best performance on validation data, is returned as the final parameters.

### 3.3 Column Specificity Meta-Learning

In ST-Semi, $\mathcal{F}_m$ is the column specificity meta-learning (CSML) algorithm used to update $\theta_w$ for few-shot tables. Motivated by the objects of pre-trained text-to-SQL models, for $(q, H^t)$, we define two new training objects to adopt meta-learning procedure:

- Predict whether $H^t$ exists in the SELECT clause

```plaintext
Algorithm 1 Self-Training Framework

Require: Labeled set $\mathcal{A}$, unlabeled set $\mathcal{U}$, validation set $\mathcal{D}_v$, basic model $\mathcal{M}$, hyper-parameters $\alpha, \beta, \gamma, \delta, \eta$.
1: Initialize $\mathcal{M}$ with random parameters $\theta$
2: $s \leftarrow 0$
3: while $s < \lambda$ do
4:   for batch $B \subseteq \mathcal{A}$ do
5:     Evaluate $\hat{B} = \{(q_i, T_i, y_i)\}$,
6:     $\nabla_{\theta} L_w = \nabla_{\theta} \sum_i L(\mathcal{M}(q_i, T_i, \theta), y_i)$
7:     Update parameters with gradient descent: $\theta \leftarrow \theta - \alpha \nabla_{\theta} L_w$
8:   end for
9:   $s \leftarrow \max(s, \mathcal{M}(\theta, \mathcal{D}_v))$
10: end while
11: while not done do
12:   Pseudo labeled set $\mathcal{P} \leftarrow \emptyset$
13:   for $q_i, T_j \in \mathcal{U}$ do
14:     Predict pseudo label $\hat{y}_i \leftarrow \mathcal{M}(q_j, T_j, \theta_w)$
15:     calculate confidence score $\psi_i$ for $\hat{y}_i$
16:     $\mathcal{P} \leftarrow \mathcal{P} \cup (q_j, T_j, \hat{y}_i, \psi_i)$
17:   end for
18:   Set $\psi = 1$ for $y_i \in \mathcal{A}$
19:   Sample $\mathcal{D}_{\mathcal{P}} \subseteq \mathcal{D}_{\mathcal{M}} = \mathcal{A} \cup \mathcal{P}$
20:   Meta learning training step: $\theta_m \leftarrow \mathcal{F}_m(\mathcal{D}_{\mathcal{P}}, \theta_m, \gamma, \delta, \eta)$
21:   Sample $\mathcal{P}' \subseteq \mathcal{P}$
22:   for batch $B \subseteq \mathcal{D}_m = \mathcal{A} \cup \mathcal{P}'$ do
23:     Evaluate $\hat{B} = \{(q_k, T_k, y_k, \psi_k)\}$,
24:     $\nabla_{\theta} L_b = \nabla_{\theta} \sum_k \nabla_{\mathcal{M}(q_k, T_k, \theta_m)} L_b(\mathcal{M}(q_k, T_k, \theta_m), y_k)$
25:     Update parameters with gradient descent: $\theta_m \leftarrow \theta_m - \beta \nabla_{\theta} L_b$
26:   end for
27:   $\theta_w \leftarrow \theta_m$
28: end while
29: return $\theta_w$
```
of $y$ corresponding to $q_i$. 
- Predict whether $\mathcal{H}^t$ exists in the WHERE clause of $y$ corresponding to $q_i$.

There are two reasons for using these two objects: 1) First, they are relevant to SC and WC tasks, which are fundamental for text-to-SQL in our basic model $\mathcal{M}$. 2) More importantly, their results are very dependent and sensitive to the table which means that they are susceptible to the few-shot tables. For each sample $(q, \mathcal{H}^t)$, the loss is calculated by a binary cross-entropy.

$$
L = y_{wce} \cdot \log (P_{wce}) + (1 - y_{wce}) \cdot \log (1 - P_{wce})
$$

where $y_{wce} \in \{0, 1\}$ and $y_{wce} \in \{0, 1\}$ are the gold labels. $P_{wce}$ and $P_{wce}$ are calculated by (2) and (3).

Moreover, in CSML, different $(q, \mathcal{H}^t)$ have different contributions when calculating the total loss according to the column specificity of $\mathcal{H}^t$. The quantitative calculation is as follows.

$$
\mu_{\mathcal{H}^t} = \frac{N_{total}}{N_{distinct} \cdot N_h},
$$

where $N_{total}$ denotes the total frequency of all the columns and $N_{distinct}$ denotes the distinct number of columns. $N_h$ denotes the frequency of the header $h_i$. Here, $N_{total}$ divided by $N_{distinct}$ represents the average frequency of all columns. The greater the $\mu_{\mathcal{H}^t}$, the more special of $\mathcal{H}^t$.

In task sampling, the mixed data $D_M = \{(q_i, \mathcal{H}_i, \psi_i, \mu_i)\}$ is shuffled, where $\psi_i$ is derived from the score of the sample corresponding to $q_i$. From $D_M$, each task randomly samples support set $S$ and query set $Q$ in $n_w$-way $k_s/k_c$-shot settings, where the samples related to the same table are treated as “a way”.

The training procedure is shown in Algorithm 2. For each task $T$, the parameter update is divided into two steps. In the first step, loss $L_S$ is calculated by evaluating the support set $S$ using $\psi$ and $\mu$ as weights (Line 3). $\theta_w'$ can be obtained by gradient update from $\theta_w$ with learning rate $\gamma$ (Line 4). In the second step, loss $L_Q$ is calculated by evaluating the query set $Q$ on $\theta'_w$ (Line 5). $L_S$ and $L_Q$ are joined to calculate the gradient which is used to update $\theta'_w$ to new $\theta_w$ with learning rate $\delta$ (Line 5). After all tasks are iterated, the final parameters $\theta_m$ are used as the output of the meta-learning step in ST-Semi.

4 Experiments

In this section, we compare ST-SQL to other existing methods to analyze the impact of different factors on few-shot performance.

4.1 Experiment Setup

4.1.1 Dataset

We evaluate the model performance on the following two benchmarks.

WikiSQL (Zhong et al., 2017) is currently the largest open-domain single-table text-to-SQL dataset which contains more than 20k tables collected from Wikipedia. Here, 80k questions are divided into 56,355 training questions, 8,421 development questions, and 15,878 test questions.

ESQL (Chen et al., 2021a) is a Chinese domain-specific single-table text-to-SQL dataset containing 17 tables, 10k training questions, 1,000 development questions, and 2,000 test questions. ESQL also provides zero-shot development and test set which contain 494 questions and 972 questions separately.

We also utilize WikiTableQuestions (Pasupat and Liang, 2015) as an available unlabeled data source. It is an open-domain single-table QA dataset that provides more than 2k tables and 20k question-answer pairs. In our experiments, when using the full training data of WikiSQL, all the NLQ-table pairs in WikiTableQuestions are used.

4.1.2 Evaluation Metrics

Logical Form (LF) accuracy is a general metric for text-to-SQL which evaluates the exact matching results between the predicted SQL and the gold SQL. Execution accuracy (EX) is another metric that evaluates whether query results of the predicted SQL and gold SQL are identical.

4.1.3 Implementation Details

We performed the experiments on Tesla V100 Super GPU. We used AdamW as the optimizer for training. In the experiments on WikiSQL, We did
not limit the number of column content $k$ because all tables in WikiSQL only have a small amount of data. On the contrary, the scale of the tables in ESQL is very large, so we set $k$ equal to 5 when experimenting on ESQL. The settings of the other hyper-parameters are in Table 1. Our implementation is publicly available.\footnote{https://github.com/ygxw0909/ST-SQL}

<table>
<thead>
<tr>
<th>Params</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha, \beta$</td>
<td>3e-5</td>
<td>learning rate of ST-Semi</td>
</tr>
<tr>
<td>$\gamma, \delta$</td>
<td>3e-5, 1e-5</td>
<td>learning rate of CSML</td>
</tr>
<tr>
<td>$\eta$</td>
<td>0.5</td>
<td>the weight of loss calculation in CSML</td>
</tr>
<tr>
<td>$[D_{r}]$</td>
<td>100/300</td>
<td>the number of sampled tasks at each iteration</td>
</tr>
<tr>
<td>$n_{v, k_v}$</td>
<td>4, 15, 5</td>
<td>the specific settings for each task</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>80.0/65.0</td>
<td>the threshold of warm boot</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>20%/5%</td>
<td>sampling ratio of the unlabeled data</td>
</tr>
<tr>
<td>$\zeta, \tau$</td>
<td>0.1, 2.5</td>
<td>the settings for confidence calculation</td>
</tr>
</tbody>
</table>

Table 1: Hyper-parameters settings. “X/X” denotes the experiments that use full training data or in few-shot settings separately.

### 4.1.4 Methods for Comparison

We compared with notable work that has reported results on WikiSQL task, including Seq2SQL (Zhong et al., 2017), Coarse2Fine (Dong and Lapata, 2018), SQLNet (Xu et al., 2017), TypeSQL (Yu et al., 2018a), X-SQL (He et al., 2019), IE-SQL (Ma et al., 2020), and SDSQL (Hui et al., 2019), SQLova (Hwang et al., 2019), HydraNet (Lyu et al., 2020), BRIDGE (Lin et al., 2020), TaBERT (Yin et al., 2020), GRAPPA (Yu et al., 2020), TAPAS (Herzig et al., 2020), and MC-SQL (Chen et al., 2021a).

### 4.2 Overall Results on WikiSQL

We first evaluated the performance of ST-SQL on the full set of WikiSQL. The experimental results are shown in Table 2. ST-SQL ranks the third best on both LF and EX on the test set of WikiSQL while the SSL-only setting achieves state-of-the-art results. Here, the “SSL-only” setting represents the ST-SQL that is removed the meta-learning step from ST-Semi. It proves that self-training on additional unlabeled data is useful to enhance the model, and can even perform better than using additional annotation. Moreover, the middle of Table 2 shows that semi-supervised learning is competitive with tabular pre-trained models. Here, a possible reason the drop brought by CSML is that mini-batch can optimize parameters more stably than meta-learning in the case of sufficient training data.

### 4.3 Few-Shot Test

To evaluate the performance of ST-SQL on a few-shot setting, we establish the following settings on WikiSQL and ESQL.

**Data Settings** For the training set of WikiSQL and ESQL, we sampled 1% (563) and %5 (500) NLQs-table pairs, respectively, to build the few-shot training sets. The remaining data of the training set is regarded as unlabeled data. We also consider the zero-shot settings, which is an extreme case of the few-shot setting. For WikiSQL, we sampled 2% (359) tables and all the related NLQs (1148) from the training set and the development/test set has 2281/4148 unseen tables. For ESQL, we followed the few-shot setting and directly used its released zero-shot development and test set which contain 7 unseen tables.

**Ablation Settings** To explore the contribution of each component of ST-SQL under the few-shot scenario, we applied the following settings.

- **Basic** We only employed the basic model described in Section 3.1.
- **Basic+SSL** we applied ST-Semi to the basic model but removed the meta-learning step.
- **Basic+SSL+ML** We replaced CSML with an existing meta-learning algorithm (Chen et al., 2021a) of the text-to-SQL field.

**Baseline Settings** We select three categories of methods from Table 2 for comparison in the few-shot/zero-shot test. First, SQLova (Hwang et al., 2019), HydraNet (Lyu et al., 2020), and BRIDGE (Lin et al., 2020) area supervised.

<table>
<thead>
<tr>
<th>Method</th>
<th>Dev.LF</th>
<th>Dev.EX</th>
<th>Test.LF</th>
<th>Test.EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq2SQL</td>
<td>49.5</td>
<td>60.8</td>
<td>48.3</td>
<td>59.4</td>
</tr>
<tr>
<td>SQLNet</td>
<td>63.2</td>
<td>69.8</td>
<td>61.3</td>
<td>68.0</td>
</tr>
<tr>
<td>TypeSQL</td>
<td>68.0</td>
<td>74.5</td>
<td>66.7</td>
<td>73.5</td>
</tr>
<tr>
<td>Coarse2Fine</td>
<td>72.5</td>
<td>79.0</td>
<td>71.7</td>
<td>78.5</td>
</tr>
<tr>
<td>SQLova</td>
<td>81.6</td>
<td>87.2</td>
<td>80.7</td>
<td>86.2</td>
</tr>
<tr>
<td>X-SQL</td>
<td>83.8</td>
<td>89.5</td>
<td>83.3</td>
<td>88.7</td>
</tr>
<tr>
<td>HydraNet</td>
<td>83.6</td>
<td>89.1</td>
<td>83.8</td>
<td>89.2</td>
</tr>
<tr>
<td>MC-SQL+</td>
<td>84.1</td>
<td>89.7</td>
<td>83.7</td>
<td>89.4</td>
</tr>
<tr>
<td>IE-SQL+</td>
<td>84.6</td>
<td>88.7</td>
<td>84.6</td>
<td>88.8</td>
</tr>
<tr>
<td>SDSQL+</td>
<td>86.0</td>
<td>91.8</td>
<td>85.6</td>
<td>91.4</td>
</tr>
<tr>
<td>BRIDGE*</td>
<td>86.2</td>
<td>91.7</td>
<td>85.7</td>
<td>91.1</td>
</tr>
<tr>
<td>TaBERT*</td>
<td>84.0</td>
<td>89.6</td>
<td>83.7</td>
<td>89.1</td>
</tr>
<tr>
<td>GRAPPA*</td>
<td>85.9</td>
<td>-</td>
<td>84.7</td>
<td>-</td>
</tr>
<tr>
<td>TAPAS*</td>
<td>85.1</td>
<td>-</td>
<td>83.6</td>
<td>-</td>
</tr>
<tr>
<td>ST-SQL*</td>
<td>85.7</td>
<td>90.8</td>
<td>85.4</td>
<td>90.3</td>
</tr>
<tr>
<td>ST-SQL(SSL-only)*</td>
<td>86.4</td>
<td>91.9</td>
<td>85.8</td>
<td>91.6</td>
</tr>
</tbody>
</table>

Table 2: Overall results on WikiSQL. "*" denotes the model using column content. "+" denotes the model is trained with additional labeling features. "-" denotes the result is based on the base version of the pretraining model.
<table>
<thead>
<tr>
<th>Method</th>
<th>FS on WikiSQL</th>
<th>ZS on WikiSQL</th>
<th>FS on ESQL</th>
<th>ZS on ESQL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dev.LF Test.LF</td>
<td>Dev.LF Test.LF</td>
<td>Dev.LF Test.LF</td>
<td>Dev.LF Test.LF</td>
</tr>
<tr>
<td>SQLova</td>
<td>35.7 35.1</td>
<td>46.2 45.7</td>
<td>59.1 54.9</td>
<td>52.0 53.3</td>
</tr>
<tr>
<td>MC-SQL*</td>
<td>52.8 53.7</td>
<td>62.8 62.8</td>
<td>62.3 63.0</td>
<td>53.0 53.9</td>
</tr>
<tr>
<td>HydraNet</td>
<td>66.3 66.0</td>
<td>68.7 68.9</td>
<td>66.8 68.8</td>
<td>63.6 66.8</td>
</tr>
<tr>
<td>BRIDGE*</td>
<td>64.4 64.0</td>
<td>70.3 70.5</td>
<td>- -</td>
<td>- -</td>
</tr>
<tr>
<td>TaBERT+SQLova*</td>
<td>56.1 55.9</td>
<td>72.9 72.9</td>
<td>- -</td>
<td>- -</td>
</tr>
<tr>
<td>TAPAS+basic*</td>
<td>43.4 42.2</td>
<td>63.1 62.8</td>
<td>- -</td>
<td>- -</td>
</tr>
<tr>
<td>GRAPPA+basic*</td>
<td>74.1 74</td>
<td>76.2 76.2</td>
<td>- -</td>
<td>- -</td>
</tr>
<tr>
<td>ST-SQL*</td>
<td>78.1 78.9</td>
<td>79.1 79.0</td>
<td>77.1 76.1</td>
<td>71.1 71.1</td>
</tr>
<tr>
<td>Basic*</td>
<td>71.5 71.5</td>
<td>74.1 73.5</td>
<td>67.2 70.5</td>
<td>66.8 68.6</td>
</tr>
<tr>
<td>Basic+SSL*</td>
<td>77.2 77.7</td>
<td>78.3 77.9</td>
<td>69.5 69.7</td>
<td>67.6 69.4</td>
</tr>
<tr>
<td>Basic+SSL+ML*</td>
<td>76.2 76.2</td>
<td>77.7 77.1</td>
<td>72.2 72.8</td>
<td>68.2 69.2</td>
</tr>
</tbody>
</table>

Table 3: Results of the few-shot and zero-shot tests. "*" denotes the model using column content.

The results in Table 3 show that our method outperforms all the baselines and gains significant improvements on few-shot and zero-shot tests on both WikiSQL and ESQL. These representative end-to-end methods can not handle few-shot and zero-shot tables very well. For the pre-trained models, we set different basic models according to their output characteristics. However, the Chinese version of these pre-training models had not been released yet, so that they were only tested on WikiSQL. Since they involve a wealth of prior knowledge and do not require many samples to fine-tune, they got better performance on WikiSQL, especially GRAPPA. Despite that, the proposed ST-Semi framework demonstrated a stronger ability to effectively using on additional unlabeled data and deal with few-shot and zero-shot tables. Amazingly, the accuracy of ST-SQL on WikiSQL with only a 1% labeling rate is close to some methods using full labeled training data.

In the ablation tests, first, the basic ST-SQL model showed the significance of table content according to the comparison with HydraNet. The semi-supervised learning in ST-Semi also improved the accuracy greatly on WikiSQL. However, it did not contribute obviously to ESQL, the reason we supposed is that, as a domain-specific dataset, ESQL is more difficult than WikiSQL to be adapted, and it leads to much noise of pseudo labels to semi-supervised learning without the assistance of meta-learning. While it was combined with CSML as ST-SQL, the promotion becomes remarkable. For comparing our CSML with other similar meta-learning algorithms, we replaced CSML with the training process in (Chen et al., 2021a), and the results showed CSML performed much better.

4.4 Detail Analysis

Effect of ST-Semi

Figure 4 shows the impact of the adoption of ST-Semi on the trend of accuracy changes during the training process in the few-shot tests. After reaching the threshold and finishing warm boot, ST-Semi further boosts the increase of accuracy compared with the basic model. The self-training process mostly completed in 30 epochs.

Moreover, we made a comparative experiment on the influence of the number of unlabeled data on ST-Semi, where the sampling ratio of mini-batch and the relevant settings of task sampling had been
scaled suitably. The results in Figure 5 show that with proper sampling-related hyper-parameter settings to ensure that unlabeled data is not excessively added to each round of training, for ST-Semi, increasing the total number of additional unlabeled data can improve the effectiveness of the ST-SQL.

![Figure 5: Results of using different sizes of additional unlabeled sets.](image)

**Error Analysis** We randomly sampled 200 bad cases after the few-shot tests and analyzed the error reason. Here, we divide error into 5 categories and statistics the frequency of each type of error in sampled bad cases which is shown in Figure 6. According to the results, Column selection and aggregation function prediction are the tasks that still need to be promoted.

![Figure 6: Percentage of error causes.](image)

5 Related Work

Single table text-to-SQL task recently attract lots of attention since the release of WikiSQL (Zhong et al., 2017). Previous work can be mainly divided into two directions. The earlier methods (Dong and Lapata, 2018; Krishnamurthy et al., 2017; Sun et al., 2018; Zhong et al., 2017) use sequence-to-sequence models to generate SQL token by token, which is called the generation-based method. But the sketch-based methods first proposed by SQLNet (Xu et al., 2017) later show better performance. Based on that, TypeSQL (Yu et al., 2018a) adds data type of columns into representation. SQLova (Hwang et al., 2019) and X-SQL (He et al., 2019) leverage BERT (Devlin et al., 2019) and MT-DNN (Liu et al., 2019) to text-to-SQL models and achieve significant improvement. Furthermore, HydraNet (Lyu et al., 2020) proposes a column-wise model which performs better than all the former question-wise models. BRIDGE (Lin et al., 2020) designs a sequential text-DB encoder to enhance the representation which achieves outstanding performance. IE-SQL (Ma et al., 2020) transfers the task into an information extraction problem. In the latest work SDSQL (Hui et al., 2021) designs an assistant task to help model learning the dependency between questions and schema.

Recently, lots of customized pre-training models appear in different research areas, as well as text-to-SQL. TaBERT (Yin et al., 2020) is proposed for a joint understanding of textual and tabular data, where table content is leveraged into representation. TAPAS (Herzig et al., 2020) is proposed for Table QA, but the table encoding ability also can be used in text-to-SQL. GRAPPA (Yu et al., 2020) is proposed for table semantic parsing combined with a grammar-augmented pre-training framework.

Few-shot and zero-shot problem in the text-to-SQL task is first mentioned by (Chang et al., 2020), where the model is additionally trained to map the entity of questions to the columns of tables. But the limitation is that additional annotations are needed. MC-SQL (Chen et al., 2021a) leverages table content features into embedding and uses meta-learning for training. However, the experiment shows that the meta-learning algorithm is not greatly helpful.

6 Conclusion

In this paper, we presented a new self-training method on the single table text-to-SQL task. We leveraged column content into a column-wise model as the basic model. Base on that, we designed a self-training framework ST-Semi which combines semi-supervised learning and meta-learning technique to make effective use of unlabeled data and adopt self-training ability. Moreover, we proposed a column specificity meta-learning algorithm to handle few-shot tables. The experimental results indicated that our method outperforms the other baselines on multiple benchmarks in few-shot settings. Furthermore, our method also showed a good performance on zero-shot tables. In future work, we will try to optimize the design of meta-learning and expand the research area to multi-tables, complex text-to-SQL.
References


Table 4: The results of each sub-task in few-shot and zero-shot tests on WikiSQL.

<table>
<thead>
<tr>
<th>Method</th>
<th>SC</th>
<th>SA</th>
<th>WN</th>
<th>WC</th>
<th>WO</th>
<th>WV</th>
<th>SC</th>
<th>SA</th>
<th>WN</th>
<th>WC</th>
<th>WO</th>
<th>WV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLova</td>
<td>81.9</td>
<td>80.8</td>
<td>77.3</td>
<td>77.4</td>
<td>88.8</td>
<td>88.4</td>
<td>66.1</td>
<td>66.3</td>
<td>77.6</td>
<td>77.4</td>
<td>67.0</td>
<td>66.9</td>
</tr>
<tr>
<td>MC-SQL*</td>
<td>82.2</td>
<td>81.8</td>
<td>83.8</td>
<td>85.2</td>
<td>92.2</td>
<td>91.2</td>
<td>86.3</td>
<td>85.3</td>
<td>82.2</td>
<td>81.8</td>
<td>73.8</td>
<td>74.0</td>
</tr>
<tr>
<td>HydraNet</td>
<td>94.2</td>
<td>91.1</td>
<td>86.2</td>
<td>86.7</td>
<td>92.9</td>
<td>91.2</td>
<td>84.0</td>
<td>83.1</td>
<td>89.5</td>
<td>88.9</td>
<td>85.4</td>
<td>84.1</td>
</tr>
<tr>
<td>BRIDGE*</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SQLova*</td>
<td>88.2</td>
<td>88.1</td>
<td>88.2</td>
<td>88.5</td>
<td>92.2</td>
<td>91.2</td>
<td>86.3</td>
<td>85.3</td>
<td>82.2</td>
<td>81.8</td>
<td>73.8</td>
<td>74.0</td>
</tr>
<tr>
<td>MC-SQL</td>
<td>91.4</td>
<td>90.9</td>
<td>87.9</td>
<td>88.5</td>
<td>95.7</td>
<td>95.1</td>
<td>92.0</td>
<td>90.8</td>
<td>88.1</td>
<td>87.5</td>
<td>86.3</td>
<td>80.0</td>
</tr>
<tr>
<td>HydraNet*</td>
<td>95.8</td>
<td>95.5</td>
<td>88.0</td>
<td>88.4</td>
<td>94.0</td>
<td>93.5</td>
<td>84.9</td>
<td>84.4</td>
<td>91.7</td>
<td>91.1</td>
<td>86.0</td>
<td>85.5</td>
</tr>
</tbody>
</table>

Table 5: The results of each sub-task in few-shot and zero-shot tests on ESQL.

<table>
<thead>
<tr>
<th>Method</th>
<th>SC</th>
<th>SA</th>
<th>WN</th>
<th>WC</th>
<th>WO</th>
<th>WV</th>
<th>SC</th>
<th>SA</th>
<th>WN</th>
<th>WC</th>
<th>WO</th>
<th>WV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLova</td>
<td>86.7</td>
<td>89.7</td>
<td>96.8</td>
<td>95.9</td>
<td>98.4</td>
<td>96.7</td>
<td>78.6</td>
<td>69.8</td>
<td>89.4</td>
<td>86.4</td>
<td>75.0</td>
<td>69.3</td>
</tr>
<tr>
<td>MC-SQL*</td>
<td>90.1</td>
<td>90.3</td>
<td>94.1</td>
<td>91.7</td>
<td>97.7</td>
<td>96.1</td>
<td>81.9</td>
<td>79.7</td>
<td>80.1</td>
<td>81.4</td>
<td>72.9</td>
<td>71.3</td>
</tr>
<tr>
<td>HydraNet</td>
<td>93.0</td>
<td>97.8</td>
<td>92.6</td>
<td>93.3</td>
<td>97.5</td>
<td>96.8</td>
<td>91.3</td>
<td>93.3</td>
<td>90.2</td>
<td>92.1</td>
<td>76.5</td>
<td>80.9</td>
</tr>
<tr>
<td>ST-SQL*</td>
<td>98.9</td>
<td>98.6</td>
<td>96.0</td>
<td>97.0</td>
<td>91.1</td>
<td>97.0</td>
<td>86.3</td>
<td>86.5</td>
<td>93.5</td>
<td>93.3</td>
<td>83.0</td>
<td>81.7</td>
</tr>
<tr>
<td>SQLova*</td>
<td>88.7</td>
<td>89.7</td>
<td>96.8</td>
<td>95.9</td>
<td>98.4</td>
<td>96.7</td>
<td>78.6</td>
<td>69.8</td>
<td>89.4</td>
<td>86.4</td>
<td>75.0</td>
<td>69.3</td>
</tr>
<tr>
<td>MC-SQL</td>
<td>90.1</td>
<td>90.3</td>
<td>94.1</td>
<td>91.7</td>
<td>97.7</td>
<td>96.1</td>
<td>81.9</td>
<td>79.7</td>
<td>80.1</td>
<td>81.4</td>
<td>72.9</td>
<td>71.3</td>
</tr>
<tr>
<td>HydraNet*</td>
<td>93.0</td>
<td>97.8</td>
<td>92.6</td>
<td>93.3</td>
<td>97.5</td>
<td>96.8</td>
<td>91.3</td>
<td>93.3</td>
<td>90.2</td>
<td>92.1</td>
<td>76.5</td>
<td>80.9</td>
</tr>
</tbody>
</table>

A Appendices

Table 4 and Table 5 show the detailed results of the few-shot and zero-shot experiments on WikiSQL and ESQL. Here, we evaluated the performance of each sub-task. ST-SQL achieved the best performance for most of the sub-tasks, especially SC and WC. It reveals that CSML can bring improvements in column selection. Leveraging the column content also significantly improved the effect of WV. Benefiting from the enhancement of the ST-Semi using additional unlabeled data, all sub-tasks had a certain improvement compared to the basic model.