Improving Few-Shot Text-to-SQL with Meta Self-Training via Column Specificity

Xinnan Guo[1], **Yongrui Chen**[1], **Guilin Qi**[1*], **Tianxing Wu**[1] and **Hao Xu**[2]
[1]School of Computer Science and Engineering, Southeast University, Nanjing, China
[2]Zhejiang Lab, Zhejiang, China
guoxinnan0727@163.com, {yrchen, gqi, tianxingwu}@seu.edu.cn, xuh@zhejianglab.com

Abstract

The few-shot problem is an urgent challenge for single-table text-to-SQL. Existing methods ignore the potential value of unlabeled data, and merely rely on a coarse-grained Meta-Learning (ML) algorithm that neglects the differences of column contributions to the optimization object. This paper proposes a Meta Self-Training text-to-SQL (MST-SQL) method to solve the problem. Specifically, MST-SQL is based on column-wise HydraNet and adopts self-training as an effective mechanism to learn from readily available unlabeled samples. During each epoch of training, it first predicts pseudo-labels for unlabeled samples and then leverages them to update the parameters. A fine-grained ML algorithm is used in updating, which weighs the contribution of columns by their specificity, in order to further improve the generalizability. Extensive experimental results on both open-domain and domain-specific benchmarks reveal that our MST-SQL[1] has significant advantages in few-shot scenarios, and is also competitive in standard supervised settings.

1 Introduction

Text-to-SQL is a popular semantic parsing task, which aims to translate natural language questions (NLQ) into SQL programs to realize intelligent access to databases. As the most fundamental setup, single-table text-to-SQL has provided the foundation of many real-world applications, such as financial markets [Sun *et al.*, 2020] and electric energy [Chen *et al.*, 2021]. In this task, which is illustrated in Figure 1, all the target SQL programs follow a unified skeleton (golden box), thereby SQL generation can be decomposed into multiple sub-tasks for predicting components. Benefiting from tabular pre-trained models [Yin *et al.*, 2020; Herzig *et al.*, 2020; Yu *et al.*, 2021] and encoder-subtask frameworks [Hwang *et al.*, 2019; Lyu *et al.*, 2020], existing methods achieve exciting results on the open-domain benchmark [Zhong *et al.*, 2017].

However, most of them require large-scale SQL annotations as the supervision signals for training. In fact, obtaining such

NLQ: *How many banking companies have headquarters in the USA?*

Company	*Headquarters*	*Industry*	…	*Profits*	*Market Value*
…	…	…	…	…	…
Citigroup	USA	Banking	…	21.54	247.42
HSBC	UK	Banking	…	16.63	202.29
Toyota	Japan	Automotive	…	11.68	217.69
…	…	…	…	…	…
ExxonMobil	USA	Oil and gas	…	39.50	410.65

```
SELECT COUNT(Company)
WHERE Headquarters = "USA"
    AND Industry = "Banking"
```

SQL Skeleton

```
SELECT $AGG $SCOL
WHERE $WCOL¹ $OP¹ $VALUE¹
    (AND $WCOLⁱ $OPⁱ $VALUEⁱ)*
```

Figure 1: An example of single-table text-to-SQL.

rich labeled training data is not only costly, but also infeasible in many sensitive user applications due to data access and privacy constraints. In real-world scenarios, the training-used NLQ-SQL pairs for each table are often insufficient, which consequently results in over-fitting. Some works [Chang *et al.*, 2020; Chen *et al.*, 2021] call it a few-shot problem and demonstrate that it is the main bottleneck of single-table text-to-SQL technology from theory to application.

In a few previous attempts, [Chang *et al.*, 2020] deals with the problem by using an auxiliary mapping task while still suffering from the high cost of manually labeled annotations; [Chen *et al.*, 2021] leverages a promising meta-learning (ML) algorithm to improve the fast adaption capability of the model. Unfortunately, the algorithm applies a one-size-fits-all updating step for all columns while neglecting the difference of their contributions, thus improving little to the task.

Self-training [Scudder, 1965] to exploit unannotated utterances provides a solution to the few-shot problem in many fields [Li *et al.*, 2019; Qi *et al.*, 2020; Wei *et al.*, 2021]. For each table, unlabeled NLQs can improve model generalizability and ease over-fitting caused by limited labeled training data. During the online iteration of a text-to-SQL system, these NLQs are readily available from previous user records. In addition, we hypothesize that distinct columns contribute differently to optimization. For example, the column "Market Value" almost merely appears in financial related tables while "Date" is more common for most tables. Intuitively, to learn the generic knowledge, the object should focus on the common columns, which are closer to the center of the sample space.

Motivated by these, this paper proposes a new method, Meta Self-Training text-to-SQL (MST-SQL), to solve the few-shot problem. Initially, to separately count the contribution of

[1]Code available at https://github.com/ygxw0909/MST-SQL
* Contact Author

each column to the optimization, we select Hydranet [Lyu *et al.*, 2020], a strong baseline on WikiSQL with column-wise inputs, as our basic model. Thereafter, we propose a *meta self-training* framework to optimize the basic model by utilizing only a few labeled data and unlabeled NLQs. Concretely, during each epoch of training, the model first predicts the SQL programs for each unlabeled NLQ as their pseudo labels, and then updates its parameters with the loss of both the labeled data and sampled pseudo labeled data. To further improve the generalizability of the model, during the updating, we adopt a fine-grained meta-learning algorithm to guide the optimization direction. On the one hand, the algorithm is merely applied to optimize the objects directly relevant to column prediction because they are the fundamental of the whole task. On the other hand, it weighs each column by its specificity, to make the common columns contribute more greatly to the total loss. The contributions of this paper can be summarized as:

- We propose a new text-to-SQL method that performs self-training with readily available unlabeled data. To the best of our knowledge, it is the first time to leverage self-training to solve few-shot text-to-SQL.
- We propose a column specificity meta-learning algorithm for optimization, which makes the model focus on common columns to learn the generic knowledge.
- We conduct comprehensive experiments on the open-domain benchmark WikiSQL and domain-specific benchmark ESQL [Chen *et al.*, 2021]. Our method outperforms all the baselines on few-shot tests and also maintains competitive results in the rich-resource scenario.

2 Preliminaries

Given an NLQ q and a table $\mathcal{T} = \{(h^1, \mathcal{C}^1), ..., (h^m, \mathcal{C}^m)\}$, where m is the number of the columns, h^i is the i-th column header, and $\mathcal{C}^i$ is the set of cells under h^i, the goal of single-table text-to-SQL is to return a SQL query y. Each y follows a unified skeleton in Figure 1, where the tokens with $ indicate slots to be filled and * represents zero or more times.

Following previous work [Hwang *et al.*, 2019; Lyu *et al.*, 2020], we decompose the task into six sub-tasks: 1) **Select-Column (SC)**: Predicting `$SCOL`, the column in the SELECT clause. 2) **Select-Aggregation (SA)**: Predicting `$AGG`, the aggregation function in the SELECT clause. 3) **Where-Number (WN)**: Predicting $\mathcal{N}$, the number of conditions in the WHERE clause. 4) **Where-Column (WC)**: Predicting $\texttt{\$WCOL}^i$, the column in the i-th condition. 5) **Where-Operator (WO)**: Predicting $\texttt{\$OP}^i$, the operator of i-th condition. 6) **Where-Value (WV)**: Extract $\texttt{\$VALUE}^i$ from q, which is the value of i-th condition. In all the sub-tasks, SC and WC are most fundamental because they are directly related to table schema and other sub-tasks need to be solved by their results. Previous studies [Lyu *et al.*, 2020; Yin *et al.*, 2020] have proved that SC and WC play the most important roles for SQL generation.

Our work considers the following scenario for self-training: the training data is denoted by $\mathcal{D} = (\mathcal{A}, \mathcal{U})$. Here, $\mathcal{A} = \{a^1, a^2, ..., a^{|\mathcal{A}|}\}$ is the set of labeled data, where $a^i = (q^i, \mathcal{T}^i, y^i)$ has a gold SQL label y^i. $\mathcal{U} = \{u^1, u^2, ..., u^{|\mathcal{U}|}\}$ is the set of unlabeled data, where $u^i = (q^i, \mathcal{T}^i)$.

3 Method

3.1 Overview

Figure 2 illustrates the overview of our proposed MST-SQL, which wraps a basic text-to-SQL model $\mathcal{M}_\theta$ with a meta self-training framework $\mathcal{F}$. Following previous work [Hwang *et al.*, 2019; Lyu *et al.*, 2020], $\mathcal{M}_\theta$ adopts an encoder-subtask architecture to generate SQL, $y = \mathcal{M}_\theta(q, \mathcal{T})$, where θ denotes the model parameters. To break its bottleneck in few-shot scenarios, $\mathcal{F}$ optimizes θ by learning from only a few labeled data $\mathcal{A}$ and additional unlabeled data $\mathcal{U}$,

$$\mathcal{M}_{\theta^*} = \mathcal{F}(\mathcal{M}_\theta, \mathcal{A}, \mathcal{U}) \tag{1}$$

where θ^* denotes the parameters with a strong generalizability.

3.2 Basic Model

We adopt Hydranet [Lyu *et al.*, 2020] as $\mathcal{M}_\theta$ because of its column-wise input form. Specifically, an original input $(q, \mathcal{T})$ is decomposed into m *column-input* (q, h^i). As shown in Figure 2(a), $\mathcal{M}_\theta$ consists of an encoding module and a sub-task module.

Encoding Module

This module is a pre-trained RoBERTa [Liu *et al.*, 2019], used to obtain the hidden states of each (q, h^i). In view of the improvement brought by using table contents [Chen *et al.*, 2021], we construct the input sequence of tokens, including an symbol `[CLS]`, a column type t^i, a column header h^i, table contents, and the NLQ q. Here, the contents are k cells in $\mathcal{C}^i$ with the highest literal scores [Chen *et al.*, 2021].

RoBERTa converts each input token into a hidden vector by attention mechanism. The obtained $\mathbf{h}_q^k \in \mathbb{R}^d$ denotes the vector of x_q^k, the k-th token in q, and $\mathbf{h}^i_{\texttt{[CLS]}} \in \mathbb{R}^d$ is regarded as the semantic vector of the entire column-input (q, h^i).

Sub-Task Module

This module solves the sub-tasks using the hidden vectors of all column-inputs. First, considering the fundamental SC task, all the columns are ranked by

$$P_{sc}(h^i = \texttt{\$SCOL}|q) = \text{sigmoid}(W_{sc} \cdot \mathbf{h}^i_{\texttt{[CLS]}}), \tag{2}$$

where $W_{sc} \in \mathbb{R}^d$ is a trainable parameter matrix. Then, h^i with the highest P_{sc} is taken as `$SCOL`. Likewise, $P_{wc}(h^i \in \{\texttt{\$WCOL}^j\}|q)$ for WC is calculated with the other parameter matrix $W_{wc} \in \mathbb{R}^d$, and top-$\mathcal{N}$ h^i with the highest P_{wc} are returned as $\{\texttt{\$WCOL}^j\}$. Here, the number of WHERE conditions, $\mathcal{N}$, is predicted by

$$P_{wn}(\mathcal{N}^j|q, h^i) = \text{softmax}(W_{wn}[j,:] \cdot \mathbf{h}^i_{\texttt{[CLS]}}), \tag{3}$$

$$\mathcal{N} = \arg\max_j(\sum_i \omega^i \cdot P_{wn}(\mathcal{N}^j|q, \mathcal{H}^i)), \tag{4}$$

where $W_{wn} \in \mathbb{R}^{\hat{n} \times d}$ is an affine transformation, and ω^i is the weight calculated by $\text{softmax}(W_\omega \cdot \mathbf{h}^i_{\texttt{[CLS]}})$. Then, the remaining sub-tasks can be solved with `$SCOL` and $\{\texttt{\$WCOL}^j\}$. Concretely, $\texttt{\$OP}^j$ is the operator $o^k \in O$ with the highest conditional probability, which is calculated by

$$P_{wo}(o^k|q, \texttt{\$WCOL}^j) = \text{softmax}(W_{wo}[k,:] \cdot \mathbf{h}^{\texttt{\$WCOL}^j}_{\texttt{[CLS]}}), \tag{5}$$

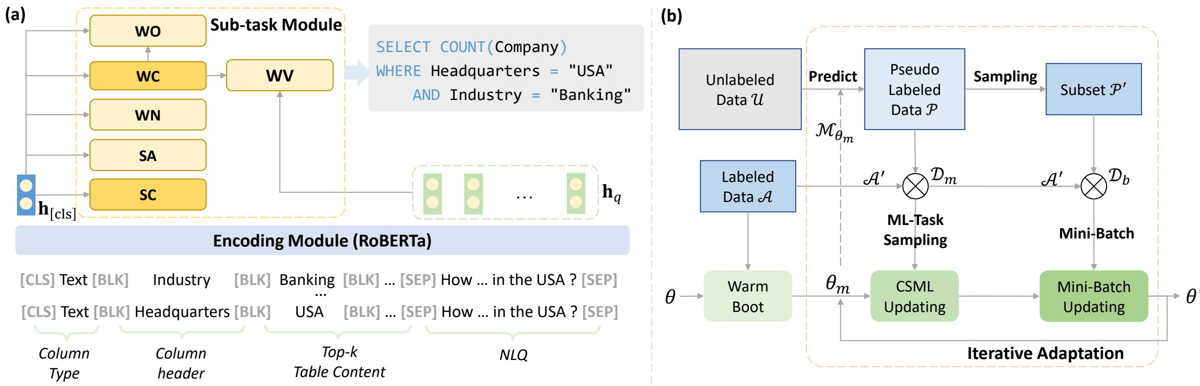

Figure 2: An overview of MST-SQL. a) The architecture of our basic model $\mathcal{M}_\theta$. b) The procedure of our meta self-training framework $\mathcal{F}$.

where $W_{wo} \in \mathbb{R}^{|O|\times d}$. Likewise, $\texttt{\$AGG}^j$ is predicted by a similar way. Afterward, x_q^k is selected as the start of $\texttt{\$VAL}^j$ for WV, which has the highest probability

$$P_{wv}^{st}(x_q^k = st|q, \texttt{\$WCOL}^j) = \text{softmax}(\mathbf{h}_q^k \cdot W_{wv}^{st}), \quad (6)$$

where $W_{wv}^{st} \in \mathbb{R}^d$, $\mathbf{h}_q^k \in \mathbb{R}^d$ is the hidden vector of x_q^k. The end index of $\texttt{\$VAL}^j$ is obtained in the same way. Finally, the SQL program is obtained by filling all the slots of the skeleton.

3.3 Meta Self-Training Framework

Algorithm 1 details our meta self-training framework $\mathcal{F}$, used to find the optimal parameters θ^* by mining the potential knowledge of unlabeled data in the following two stages.

Warm Boot

The goal of the warm boot (Line 2) is to obtain the stable parameters of $\mathcal{M}_\theta$, so as to subsequently provide the first batch of high-quality pseudo-labels. Specifically, randomly initialized $\mathcal{M}_\theta$ is trained with a conventional mini-batch strategy on labeled data $\mathcal{A}$. When the validation performance $Acc_\mathcal{V}$ reaches the defined threshold λ, stable θ_m is obtained.

Iterative Adaptation

At each epoch, $\mathcal{M}_{\theta_m}$ first predicts a SQL program $\hat{y}^i$ as a pseudo label for each unlabeled sample $u^i \in \mathcal{U}$ (Line 4). To mitigate error propagation from noisy pseudo labels, we design ψ^i to evaluate the confidence of $p^i = (q^i, \mathcal{T}^i, \hat{y}^i)$.

$$\psi^i = \sqrt[\mathcal{N}+1]{P_{sc}(\texttt{\$SCOL}|q^i) \cdot \prod_{\texttt{\$WCOL}^j} P_{wc}(\texttt{\$WCOL}^j|q^i)}, \quad (7)$$

where $P_{sc}(\texttt{\$SCOL}|q^i)$ and $P_{wc}(\texttt{\$WCOL}^j|q^i)$ are the ranking scores of predicted $\texttt{\$SCOL}$ and $\texttt{\$WCOL}^j$, respectively. Here, ψ^i merely involves SC and WC because their performances reflect the quality of the entire SQL in most cases. Finally, all the pseudo-labeled samples $(q^i, \mathcal{T}^i, \hat{y}^i, \psi^i)$ make up a pseudo-labeled set $\mathcal{P}$, and the original $\mathcal{A}$ is also wrapped to $\mathcal{A}'$, where the confidence of each sample is set to 1.

Subsequently, $\mathcal{A}' \cup \mathcal{P}$ is leveraged for back-propagation to update θ_m. Considering the priorities and characteristics of the sub-tasks, we develop a two-step updating here: (a) First, since column selection is typically table-sensitive, we propose

Algorithm 1 Meta Self-Training Framework

Require: Labeled set $\mathcal{A}$, unlabeled set $\mathcal{U}$, validation set $\mathcal{V}$, basic model $\mathcal{M}$, hyper-parameters γ, σ.
1: Initialize $\mathcal{M}_\theta$ with random parameters θ, $Acc^* \leftarrow 0$
2: Warm boot, train $\mathcal{M}_\theta$ with mini-batches on $\mathcal{A}$, obtain θ_m
3: **while** not done **do**
4: Predict pseudo labels $\hat{y}^j \leftarrow \mathcal{M}_{\theta_m}(q^j, \mathcal{T}^j)$, where $(q^j, \mathcal{T}^j) \in \mathcal{U}$, calculate confidence ψ^j for each $\hat{y}^j$, make up the pseudo labeled set $\mathcal{P}$
5: $\mathcal{A}' \leftarrow \{(q^j, \mathcal{T}^j, \hat{y}^j, 1)\}$, where each $(q^j, \mathcal{T}^j, \hat{y}^j) \in \mathcal{A}$
6: Meta training on $\mathcal{D}_m$ sampled from $\mathcal{A}' \cup \mathcal{P}$, $\theta_m \leftarrow \mathcal{F}_m(\mathcal{D}_m, \mathcal{M}_{\theta_m})$
7: $\mathcal{D}_b = \mathcal{A}' \cup \mathcal{P}'$, where $\mathcal{P}'$ is sampled from $\mathcal{P}$ with σ
8: **for** batch $\mathcal{B}$ iterated from $\mathcal{D}_b$ **do**
9: Evaluate $\mathcal{B} = \{(q^i, \mathcal{T}^i, y^i, \psi^i)\}$, $\nabla_{\theta_m}\mathcal{L}_b \leftarrow \nabla_{\theta_m} \sum_i \psi^i \mathcal{L}_b(\mathcal{M}_{\theta_m}(q^i, \mathcal{T}^i), y^i)$
10: Update parameters with gradient descent: $\theta_m \leftarrow \theta_m - \gamma \nabla_{\theta_m} \mathcal{L}_b$
11: **end for**
12: Evaluate validation accuracy $Acc_\mathcal{V}$ on $\mathcal{V}$ with $\mathcal{M}_{\theta_m}$
13: **if** $Acc_\mathcal{V} > Acc^*$ **then**
14: $\theta^* \leftarrow \theta_m$, $Acc^* \leftarrow Acc_\mathcal{V}$
15: **end if**
16: **end while**
17: **return** $\mathcal{M}_{\theta^*}$

a *Column Specificity Meta-Learning* (CSML) algorithm $\mathcal{F}_m$ to improve the most fundamental SC and WC, in order achieve the fast adaption to new tables. The training data $\mathcal{D}_m$ consists of n ML-tasks sampled from $\mathcal{A}' \cup \mathcal{P}$, which are detailed in Section 3.4. Note that only W_{sc}, and W_{wc}, and the parameters of the encoding module are updated at this step, to guide the optimization direction of the entire θ_m. (b) Thereafter, all the parameters of θ_m are updated by optimizing the total loss of all the sub-tasks with the mini-batch training (Line 7-11). Here, each batch $\mathcal{B}$ is obtained from $\mathcal{D}_b = \mathcal{A}' \cup \mathcal{P}'$, where $\mathcal{P}'$ is randomly sampled from $\mathcal{P}$ with ratio σ to prevent the excessive impact of pseudo-labels. Thus, θ_m follows the macro direction while maintaining a stable updating.

After multiple epochs, θ_m with the best performance $Acc_\mathcal{V}$ is regarded as the optimal parameter $\theta*$ (Line 12-15).

Algorithm 2 Column Specificity Meta-Learning

Require: Task set $\mathcal{D}_m = \{(\mathcal{S}, \mathcal{Q})\}$, model parameters θ_m, hyper-parameters α, β, and η
1: **for all** tasks from $\mathcal{D}_m$ **do**
2: Evaluate support set $\mathcal{S} = \{(q, h^i, y, \psi, \mu^i)\}$, $\nabla_{\theta_m}\mathcal{L}_\mathcal{S} = \nabla_{\theta_m}\sum_i \psi\mu^i\mathcal{L}^i$
3: Update parameters with gradient descent: $\tilde{\theta}_m \leftarrow \theta_m - \alpha\nabla_{\theta_m}\mathcal{L}_\mathcal{S}$
4: Evaluate query set $\mathcal{Q} = \{(q, h^j, y, \psi, \mu^j)\}$, $\nabla_{\tilde{\theta}_m}\mathcal{L}_\mathcal{Q} = \nabla_{\tilde{\theta}_m}\sum_j \psi\mu^j\mathcal{L}^j$
5: Update parameters with gradient descent: $\mathcal{L}_{\mathcal{S}+\mathcal{Q}} = \eta\mathcal{L}_\mathcal{S} + (1-\eta)\mathcal{L}_\mathcal{Q}, \theta_m \leftarrow \tilde{\theta}_m - \beta\nabla_{\tilde{\theta}_m}\mathcal{L}_{\mathcal{S}+\mathcal{Q}}$
6: **end for**
7: **return** θ_m

3.4 Column Specificity Meta-Learning

Our CSML is performed on *column-samples* to focus on the independent contribution of each column. Specifically, each original sample $(q, \mathcal{T}, y, \psi)$ is broken into column-samples (q, h^i, y, ψ) and all of them are shuffled and sampled to form n ML-tasks $\mathcal{D}_m$. The task follows a common N-way K-shot setting, i.e., N tables are involved and each table provides K related column-samples.

For each (q, h^i, y, ψ), unlike existing ML algorithms [Chen *et al.*, 2021; Wang *et al.*, 2021a], which optimize the total loss of all the sub-tasks, our CSML defines the following two binary-classification objects to improve SC and WC:

- Predict whether column h^i exists in the `SELECT` clause.
- Predict whether column h^i exists in the `WHERE` clause.

These two objects have a concise form suitable for using ML and the potential to improve the generalization ability of the model because of their table-sensitivity. In this way, the loss of each $(q, h^i, \mathcal{C}^i)$ is defined as

$$\mathcal{L}^i = H(P_{sc}(h^i|q), y_{sc}) + H(P_{wc}(h^i|q), y_{wc}) \qquad (8)$$

where $H(x, y)$ denotes the cross-entropy. y_{sc} and y_{wc} are the gold binary labels of the two objects, respectively, which can be easily obtained from SQL label y. $P_{sc}(h^i|q)$ and $P_{wc}(h^i|q)$ are the predicted probability of SC and WC, respectively.

In order to promote $\mathcal{M}_{\theta_m}$ to focus on generic knowledge, our CSML hypothesizes that common columns contribute more to the learning object, which is significantly different from the one-size-fits-all updating used in [Chen *et al.*, 2021]. Concretely, for each h^i, we define a *column specificity*

$$\mu^i = \frac{N_{distinct} \cdot N^i}{N_{total}} \qquad (9)$$

where N^i is the frequency of h_i, N_{total} is the total frequency, and $N_{distinct}$ denotes the number of distinct h_i. The equation reflects that the ratio of the frequency of h_i to the average frequency. Theoretically, the greater μ^i indicates that h^i is more common.

Our CSML is detailed in Algorithm 2. Optimizing support set loss $\mathcal{L}_\mathcal{S}$ (Line 3) first provides the possible direction of θ_m and then optimizing query set loss $\mathcal{L}_\mathcal{Q}$ (Line 5) corrects the direction for generalization. Notice that the loss $\mathcal{L}^i$ of each column-sample in CSML is re-weighted with specificity μ^i, thus common columns can lead optimization in a more general direction. In addition, the confidence ψ of each column-sample is also taken as a weight to reduce noise.

Method	Dev.LF	Dev.EX	Test.LF	Test.EX
SQLova	81.6	87.2	80.7	86.2
X-SQL	83.8	89.5	83.3	88.7
HydraNet	83.6	89.1	83.8	89.2
MC-SQL*	84.1	89.7	83.7	89.4
IE-SQL+	84.6	88.7	84.6	88.8
SeaD	84.9	90.2	84.7	90.1
SDSQL+	86.0	91.8	85.6	91.4
BRIDGE*	86.2	91.7	85.7	91.1
TaPaS*+	85.1	-	83.6	-
TaBert*+	84.0	89.6	83.7	89.1
Grappa*+	85.9	-	84.7	-
MST-SQL*+	85.7	90.8	85.4	90.3
Basic*	85.6	91.2	85.0	90.8
ST-only*+	**86.4**	**91.9**	**85.8**	**91.6**

Table 1: Results on original WikiSQL."*" denotes using table contents or accessing databases during SQL generation. "+" denotes using extra knowledge (e.g.,tabular pre-training).

4 Experiment

4.1 Experiment Setup

Dataset and Evaluation Metrics

We evaluated our MST-SQL on the two benchmarks: 1) **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 and 56,355 / 8,421 / 15,878 NLQ-SQL pairs for training / development / test. 2) **ESQL** [Chen *et al.*, 2021] is a Chinese domain-specific single-table text-to-SQL dataset containing 17 tables and 10,000 / 1,000 / 2,000 NLQ-SQL pairs for training / development / test.

We adopt Logical Form (LF) accuracy and Execution (EX) accuracy as the metrics, where the former evaluates the exact matching results between the predicted SQL and the gold one, and the latter compares the query results of the predicted SQL with the gold one.

Implementation Details

Our method ran on Tesla V100 Super GPUs. We used RoBERTa as the encoder by default. The hyper-parameters were set as follow: (1) The number of table content was set to 5. (2) The learning rate α, β, γ were set to 3e-5, 1e-5, 3e-5. (3) The weight of the loss calculation in CSML η was set to 0.5. (4) The threshold for confidence scores ζ was set to 0.1. (5) N was set to 4, and K was set to 15 / 5 for support / query set. (6) ML-task number n, sampling ratio σ, and warm-boot threshold λ were set to 80, 0.2, 80 in standard supervised setting, and the setting in few-shot tests will be discussed later.

Methods for Comparison and Ablation Settings

We compared with notable work that has reported results on WikiSQL, including X-SQL [He *et al.*, 2019], IE-SQL [Ma *et al.*, 2020], and SDSQL [Hui *et al.*, 2021], SeaD [Xuan *et al.*, 2021], SQLova [Hwang *et al.*, 2019], HydraNet [Lyu *et al.*, 2020], BRIDGE [Lin *et al.*, 2020], TaBert [Yin *et al.*, 2020],

Method	WikiSQL 1-SHOT	2-SHOT	3-SHOT	4-SHOT	ESQL 5-SHOT	10-SHOT	15-SHOT	20-SHOT
SQLova	23.3	40.8	48.7	55.3	22.3	39.6	52.0	54.7
MC-SQL	52.0	62.9	71.0	73.8	36.5	53.2	60.5	67.4
HydraNet	64.2	69.9	72.9	74.3	43.6	58.1	70.5	76.7
BRIDGE	53.6	68.9	73.1	77.3	-	-	-	-
TaBERT	57.5	67.4	71.2	72.5	-	-	-	-
Grappa	72.8	76.8	78.0	78.1	-	-	-	-
MST-SQL	**78.4**	**80.5**	**82.1**	**83.2**	**55.3**	**67.4**	**76.7**	**80.5**
Basic	69.6	74.3	76.3	77.3	45.3	60.2	72.2	77.7
ST-only	75.8	78.6	79.4	81.1	51.2	64.5	74.2	80.2

Table 2: Overall results of few-shot experiments in different shot settings of the training data. LF accuracy is used for evaluation.

Method	1-SHOT	2-SHOT	3-SHOT	4-SHOT
Basic	69.6	74.3	76.3	77.3
ST-only$_{wtq}$	72.2	75.5	76.8	77.9
ST-only$_{wiki}$	75.8	78.6	79.4	81.1
MST-SQL$_{wtq}$	72.9	77.1	77.4	78.2
MST-SQL$_{wiki}$	**78.4**	**80.5**	**82.1**	**83.2**

Table 3: Results of using different unlabeled data sources.

Grappa [Yu *et al.*, 2021], TaPaS [Herzig *et al.*, 2020], and MC-SQL [Chen *et al.*, 2021].

To evaluate the contribution of components of MST-SQL, we add two ablation settings in the subsequent experiments: 1) **Basic**: train $\mathcal{M}_\theta$ with the standard supervised learning on $\mathcal{A}$ instead of meta self-training. 2) **ST-only**: CSML step (Line 6 in Algorithm 1) is removed during the iterative adaption.

4.2 Results in Standard Text-to-SQL Setting

We first conducted the experiments in the standard text-to-SQL setting with sufficient training data. We used the full training set of WikiSQL as the labeled data $\mathcal{A}$, and toke **WikiTable-Questions** [Pasupat and Liang, 2015] as the unlabeled data $\mathcal{U}$, which provides more than 2k tables and 20k NLQs without SQL labels. Here we did not evaluate ESQL because it includes a non-negligible zero-shot challenge. The experimental results are shown in Table 1, where all the results with execution guiding (EG) [Wang *et al.*, 2018] are discarded because EG requires that the query results of SQL are always non-empty, which is unreasonable in realistic scenarios. Although the full MST-SQL ranks third on testing in terms of both LF and EX, the ST-only setting surprisingly achieves state-of-the-art results. It proves that self-training with unlabeled data can also improve the model in rich-resource scenarios. A possible reason for the drop brought by CSML is that mini-batch can optimize parameters more stably than meta-learning with sufficient training data. In addition, our MST-SQL outperforms the tabular pre-trained models (middle of Table 1) without requiring a massive text-SQL parallel corpus.

4.3 Few-Shot Tests

Compared with the standard setting, we are more concerned with the performance of text-to-SQL in real few-shot scenarios. Therefore, we reconstructed WikiSQL and ESQL to enhance the few-shot challenge. Concretely, since the original set of WikiSQL has large amounts of tables with identical schema,

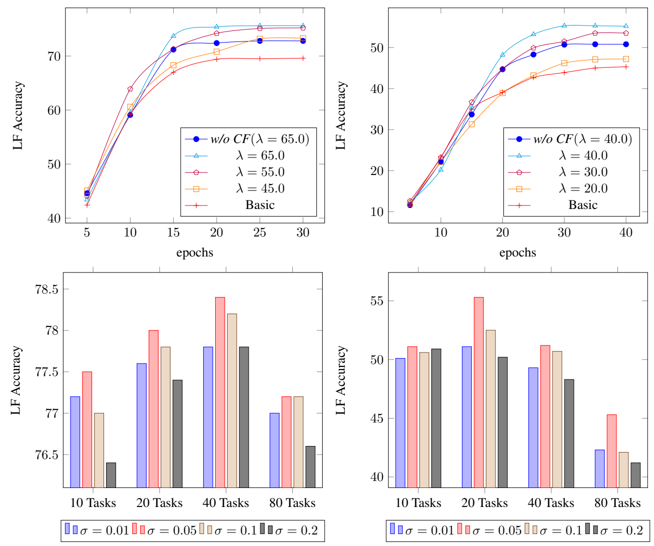

Figure 3: Comparison of the self-training process in different settings.

we merged them into approximately 9,000 new tables and randomly selected 500 of them for few-shot tests. For ESQL, we removed 2 tables of insufficient samples and retained 15 tables. For each dataset, we built four training sets by the shot number(WikiSQL: {1, 2, 3, 4}, ESQL:{5, 10, 15, 20}), and construct validation and test sets with the same setting(4 / 100 samples each table for WikiSQL / ESQL). Different from leveraging extra data (e.g., WikiTableQuestions) in the standard supervised setting, we built $\mathcal{U}$ with all the remaining NLQs of the original WikiSQL while masking their SQL annotations, in order to simulate user data collected from the same-origin tables in real-world applications.

The results are shown in Table 2. BRIDGE, TaBERT, and Grappa have no results on ESQL due to no solutions to Chinese corpus. Our MST-SQL outperforms all the compared methods with significant improvement on all shot numbers. Notably, even the state-of-the-art BRIDGE still can not solve the few-shot problem well, especially when the shot number is extremely small. Since tabular pre-trained models have learned a wealth of prior knowledge, they perform better. Compared to Grappa which pre-trained by 866.5k examples, our MST-SQL achieves better performance while utilizing less extra unlabeled data. The bottom of Table 2 indicates that both self-training and CSML contribute to the entire model. The smaller the number of shots, the greater their contribution.

	Method	SC	SA	WN	WC	WO	WV	LF
WikiSQL	MST-SQL	**98.2**	86.7	95.5	**93.7**	**94.4**	**93.5**	**78.4**
	w/o OB	97.8	87.2	**95.8**	93.0	94.0	92.5	77.3
	w/o CS	97.4	**88.2**	93.4	91.6	92.9	92.8	77.2
	w/o OB & CS	96.9	87.5	94.9	92.7	93.4	92.5	76.4
	ST-only	96.4	86.1	95.7	91.5	93.0	92.4	75.8
ESQL	MST-SQL	**93.3**	82.9	**86.5**	**80.7**	**90.0**	**87.1**	**55.3**
	w/o OB	91.5	82.2	85.5	77.1	88.1	85.7	53.1
	w/o CS	91.2	80.4	86.3	76.9	88.8	86.2	52.8
	w/o OB & CS	91.5	80.5	85.1	77.3	87.8	86.5	52.4
	ST-only	91.2	**83.9**	86.1	74.2	86.5	84.8	51.2

Table 4: Performance of sub-tasks in different CSML settings

The Impact of Unlabeled Data Source

Sources of unlabeled data may influence the entire performance. To explore the impact, we compared two types of $\mathcal{U}$ in few-shot tests: a) Using remaining samples in the same datasets as $\mathcal{U}$ (same source), which is adopted in Table 2; b) Using WikiTableQuestions as $\mathcal{U}$ (different sources). Table 3 shows the results. Both types of unlabeled data are effective due to their extra knowledge. In comparison, the unlabeled data from the same source gains a greater contribution than that of different sources. The main reason can be that the identical distribution results in less noise.

Ablation Study of Self-Training

To further explore the conditions of self-training, we trained the model with different warm-boot thresholds λ ($\{45.0, 55.0, 65.0\}$ for WikiSQL, $\{20.0, 30.0, 40.0\}$ for ESQL), and discarded the sample confidence ψ. In addition, we compared the different ML-task numbers n of $\mathcal{D}_m$ and sampling ratios σ of $\mathcal{D}_b$. Note that all the experiments here were in the hardest 1-SHOT setting for WikiSQL and 5-SHOT for ESQL. The results are illustrated in Figure 3. Lowering the threshold results in a slower convergence speed and a worse final performance. The greater effect of λ on ESQL reveals that stability is more important for domain-specific scenarios. The performance drop of *w/o CF* indicates the necessity of the confidence scores. Moreover, blindly increasing the sampling ratio and the number of ML-tasks does not always bring improvement.

Ablation Study of CSML

We also analyzed CSML to evaluate its effectiveness. The experiments follow the SHOT setting in the previous section. Specifically, we compared three settings: a) *w/o OB* denotes using the unified CSML for all the sub-tasks instead of the defined objects in Section 3.4. b) *w/o CS* means that samples are not weighted with the column specificity μ. c) *w/o OB & CS* represents deleting both objects and μ, i.e., the ML algorithm proposed in [Chen *et al.*, 2021]. Table 4 shows the performance of all the sub-tasks in different CSML settings. MST-SQL equipped with all the components achieves best on overall LF and most sub-tasks. Abandoning the new objects causes drops in almost all the sub-tasks, which reflects the importance of the adaptation to the sub-task differences. In addition, column specificity not only benefits SC and WC but also contributed to other sub-tasks. It can be due to improvements in SC and WC that raise the caps of other sub-tasks. Benefiting from the above improvements, our CSML gains better results for few-shot text-to-SQL than the existing ML algorithm [Chen *et al.*, 2021].

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; 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 method first proposed by SQLNet [Xu *et al.*, 2017] shows a better performance and is followed by many later works [Hwang *et al.*, 2019; He *et al.*, 2019; Lyu *et al.*, 2020]. Latest end-to-end works propose to use additional annotated features [Ma *et al.*, 2020], auxiliary tasks [Hui *et al.*, 2021; Xuan *et al.*, 2021], and special decoding mechanisms [Wang *et al.*, 2018; Lin *et al.*, 2020] to further improve the effectiveness of standard supervised learning.

In recent years, tabular pre-trained models have been proposed and released. TaBERT [Yin *et al.*, 2020] presents 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 can also be used in text-to-SQL. Grappa [Yu *et al.*, 2021] combines table semantic parsing with a grammar-augmented pre-training framework.

The few-shot problem in text-to-SQL is first mentioned by [Chang *et al.*, 2020], which proposes a mapping auxiliary task to enhanced the model while still suffers from the cost of additional annotations. MC-SQL [Chen *et al.*, 2021] handles the problem by a coarse-grained ML algorithm. However, its improvement is not obvious because of its one-size-fits-all updating for all the columns. Similarly, the ML used in DGMAML [Wang *et al.*, 2021a] is also coarse-grained and relies on multi-domain scenarios.

Self-training [Scudder, 1965] is promising for few-shot problems [Li *et al.*, 2019; Wei *et al.*, 2021; Qi *et al.*, 2020]. Recently, some methods [Wang *et al.*, 2021b; Hu *et al.*, 2021] integrate meta-learning to it to further improve the generalization capability. The most significant difference between these methods and our MST-SQL is that they only treat ML as a tool to evaluate the sample confidence while we use column specificity to guide the ML process and optimize the model for fast adaption to new tables.

6 Conclusion

In this paper, we presented a new meta self-training method for the few-shot problem of single-table text-to-SQL. The proposed self-training process helps the model learn the generic knowledge from readily available unlabeled data. At each epoch, the model first predicts pseudo-labels and then performs a two-step updating to optimize the parameters. A fine-grained meta-learning algorithm is employed to re-weigh the columns by their specificity in order to make sure that the common columns have a greater contribution to the optimization objects. The experimental results on both open-domain and domain-specific benchmarks indicate that our method provides a promising way for few-shot text-to-SQL. In future work, we will try to expand the proposed method to apply to the joint queries of multiple tables, in order to handle more complex text-to-SQL tasks.

Acknowledgements

This work is supported by the NSFC (Grant No. U21A20488, 62006040), the Project for the Doctor of Entrepreneurship and Innovation in Jiangsu Province (Grant No. JSSCBS20210126), the Fundamental Research Funds for the Central Universities, and ZhiShan Young Scholar Program of Southeast University.

References

[Chang *et al.*, 2020] Shuaichen Chang, Pengfei Liu, Yun Tang, Jing Huang, Xiaodong He, and Bowen Zhou. Zero-shot text-to-sql learning with auxiliary task. In *AAAI*, pages 7488–7495, 2020.

[Chen *et al.*, 2021] Yongrui Chen, Xinnan Guo, Chaojie Wang, Jian Qiu, Guilin Qi, Meng Wang, and Huiying Li. Leveraging table content for zero-shot text-to-sql with meta-learning. In *AAAI*, pages 3992–4000, 2021.

[Dong and Lapata, 2018] Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *ACL*, pages 731–742, 2018.

[He *et al.*, 2019] Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. X-SQL: reinforce schema representation with context. *CoRR*, abs/1908.08113, 2019.

[Herzig *et al.*, 2020] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. In *ACL*, pages 4320–4333, 2020.

[Hu *et al.*, 2021] Xuming Hu, Chenwei Zhang, Fukun Ma, Chenyao Liu, Lijie Wen, and Philip S. Yu. Semi-supervised relation extraction via incremental meta self-training. In *Findings of EMNLP*, pages 487–496, 2021.

[Hui *et al.*, 2021] Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, and Xiaodan Zhu. Improving text-to-sql with schema dependency learning. *CoRR*, abs/2103.04399, 2021.

[Hwang *et al.*, 2019] Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. A comprehensive exploration on wikisql with table-aware word contextualization. *CoRR*, abs/1902.01069, 2019.

[Li *et al.*, 2019] Xinzhe Li, Qianru Sun, Yaoyao Liu, Qin Zhou, Shibao Zheng, Tat-Seng Chua, and Bernt Schiele. Learning to self-train for semi-supervised few-shot classification. In *NeurIPS*, pages 10276–10286, 2019.

[Lin *et al.*, 2020] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. In *Findings of EMNLP*, volume EMNLP, pages 4870–4888, 2020.

[Liu *et al.*, 2019] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[Lyu *et al.*, 2020] Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. Hybrid ranking network for text-to-sql. *CoRR*, abs/2008.04759, 2020.

[Ma *et al.*, 2020] Jianqiang Ma, Zeyu Yan, Shuai Pang, Yang Zhang, and Jianping Shen. Mention extraction and linking for SQL query generation. In *EMNLP*, pages 6936–6942, 2020.

[Pasupat and Liang, 2015] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *ACL*, pages 1470–1480, 2015.

[Qi *et al.*, 2020] Qi Qi, Xiaolu Wang, Haifeng Sun, Jingyu Wang, Xiao Liang, and Jianxin Liao. A novel multi-task learning framework for semi-supervised semantic parsing. *IEEE TASL*, 28:2552–2560, 2020.

[Scudder, 1965] H. J. Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Trans. Inf. Theory*, 11(3):363–371, 1965.

[Sun *et al.*, 2020] Ningyuan Sun, Xuefeng Yang, and Yunfeng Liu. Tableqa: a large-scale chinese text-to-sql dataset for table-aware SQL generation. *CoRR*, abs/2006.06434, 2020.

[Wang *et al.*, 2018] Chenglong Wang, Po-Sen Huang, Alex Polozov, Marc Brockschmidt, and Rishabh Singh. Execution-guided neural program decoding. *CoRR*, abs/1807.03100, 2018.

[Wang *et al.*, 2021a] Bailin Wang, Mirella Lapata, and Ivan Titov. Meta-learning for domain generalization in semantic parsing. In *NAACL-HLT*, pages 366–379, 2021.

[Wang *et al.*, 2021b] Yaqing Wang, Subhabrata Mukherjee, Haoda Chu, Yuancheng Tu, Ming Wu, Jing Gao, and Ahmed Hassan Awadallah. Meta self-training for few-shot neural sequence labeling. In *KDD*, pages 1737–1747, 2021.

[Wei *et al.*, 2021] Colin Wei, Kendrick Shen, Yining Chen, and Tengyu Ma. Theoretical analysis of self-training with deep networks on unlabeled data. In *ICLR*, 2021.

[Xu *et al.*, 2017] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436, 2017.

[Xuan *et al.*, 2021] Kuan Xuan, Yongbo Wang, Yongliang Wang, Zujie Wen, and Yang Dong. Sead: End-to-end text-to-sql generation with schema-aware denoising. *CoRR*, abs/2105.07911, 2021.

[Yin *et al.*, 2020] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. In *ACL*, pages 8413–8426, 2020.

[Yu *et al.*, 2021] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. Grappa: Grammar-augmented pre-training for table semantic parsing. In *ICLR*. OpenReview.net, 2021.

[Zhong *et al.*, 2017] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.