Semantic Enhanced Text-to-SQL Parsing via Iteratively Learning Schema Linking Graph

Aiwei Liu
Tsinghua University
Beijing, China
liuaw20@mails.tsinghua.edu.cn

Xuming Hu
Tsinghua University
Beijing, China
hxm19@mails.tsinghua.edu.cn

Li Lin
Tsinghua University
Beijing, China
lin-l16@mails.tsinghua.edu.cn

Lijie Wen*
Tsinghua University
Beijing, China
wenlj@tsinghua.edu.cn

ABSTRACT

The generalizability to new databases is of vital importance to Text-to-SQL systems which aim to parse human utterances into SQL statements. Existing works achieve this goal by leveraging the exact matching method to identify the lexical matching between the question words and the schema items. However, these methods fail in other challenging scenarios, such as the synonym substitution in which the surface form differs between the corresponding question words and schema items. In this paper, we propose a framework named ISESL-SQL to iteratively build a semantic enhanced schema-linking graph between question tokens and database schemas. First, we extract a schema linking graph from PLMs through a probing procedure in an unsupervised manner. Then the schema linking graph is further optimized during the training process through a deep graph learning method. Meanwhile, we also design an auxiliary task called graph regularization to improve the schema information mentioned in the schema-linking graph. Extensive experiments on three benchmarks demonstrate that ISESL-SQL could consistently outperform the baselines and further investigations show its generalizability and robustness.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**;
• **Information systems** → **Structured Query Language**.

KEYWORDS

Text-to-SQL, Graph neural networks, Model Robustness

ACM Reference Format:
Aiwei Liu, Xuming Hu, Li Lin, and Lijie Wen. 2022. Semantic Enhanced Text-to-SQL Parsing via Iteratively Learning Schema Linking Graph. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539294

*Corresponding author

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539294

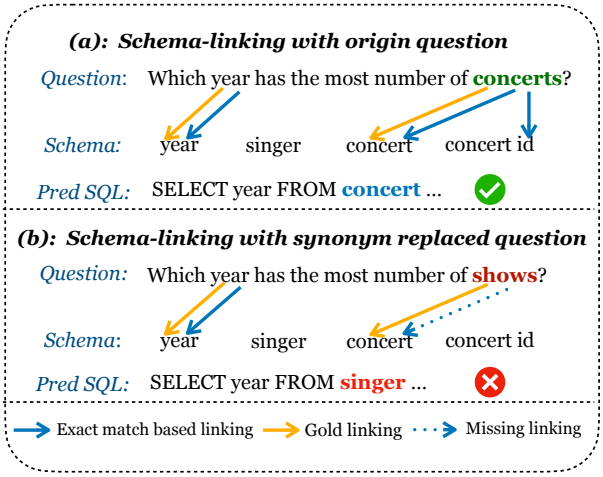

Figure 1: Examples of exact match based schema linking (EMSL). (a): EMSL works fine in the original question; (b): EMSL fails to identify table *concert* in the question with synonym substitution.

1 INTRODUCTION

As a vast amount of real-world information is stored in databases, using natural language (e.g. English) to inquire information from tabular data is of great importance for modern retrieval applications (e.g., web search engine). By translating natural language to executable SQL statements, Text-to-SQL becomes an effective solution to achieve this goal [23]. Due to databases in the real world being diverse and heterogeneous, the ability for Text-to-SQL systems to be adaptive to different databases instead of being designed for a specific database is of vital importance. In the single domain setting, the train and test data share the same database, which makes it easy for Text-to-SQL models to learn the relationship between questions and schemas (columns and tables). In contrast to that, databases in train and test data are separated under the cross domain setting, which is challenging for the Text-to-SQL models to infer the schema information mentioned in the question.

Recently, numerous cross-domain Text-to-SQL methods have been proposed, such as IRNet [17], RAT-SQL [38], LGESQL [4]. To better extract question-related schema information from databases,

all these models use schema linking as the key module to link the tokens in the question to the correct mentioned schema items and thus construct the schema linking graph. As illustrated in Figure 1 (a), a linking is established between question token *concert* and schema item *concerts* to help the Text-to-SQL model find the correct table in the predicted SQL statement.

However, these methods may fail in real-world settings where schema items are expressed in their synonym form or domain knowledge is required to obtain schema items. As shown in Figure 1 (b), when the mention *concerts* is expressed in its synonym form *shows*, the schema linking module fails to detect the correct table *concert*, which leads to the wrong generated SQL statement. To avoid the limitation of the exact matching based schema linking method, several works explored other methods. Dong et al. [11] leverage the implicit supervision from SQL queries to guide the schema linking training by reinforcement learning, which ignores the semantic information and may still fail in synonym substitution settings. To better capture the semantic relationship between question and schema, Liu et al. [30] train the schema linking module under the supervision of pseudo alignment between token and schema items from PLMs. However, the supervision from PLMs could be noisy or incomplete.

To obtain more precise and complete semantic linking between questions and schema items, we propose a novel framework named ISESL-SQL to **i**teratively build a **s**emantic **e**nhanced **s**chema-**l**inking graph from PLMs during the Text-to-**SQL** training process. We formulate the schema linking procedure as graph construction in which schema linking edges are established between question nodes and schema nodes. Our ISESL-SQL framework introduces three different modules to construct schema linking graph: initial graph probing module, implicit graph learning module and graph regularization module. To be robust in challenging settings such as synonym substitution, the initial graph probing module constructs the initial semantic schema linking graph by masking one question token per time and observing how the PLM embeddings of the schema items are affected.

Because the initial graph is not optimal for the downstream Text-to-SQL task, learning an adaptive graph structure during model training is also needed. Therefore, the implicit graph learning module learns to generate a sparse implicit weighted graph during training and the whole process is optimized through the downstream Text-to-SQL task in each iteration, which could be viewed as iterative refinement. The implicit weighted graph is then combined with the initial schema linking graph to obtain the schema linking graph in each iteration. We further apply a modified version of relational graph attention network (RGAT) on it to learn a joint embedding of question and schema nodes.

Although the combined schema linking graph could be optimized by the downstream Text-to-SQL task, whether the model learns the standard schema linking graph is not guaranteed. Despite the minimal impact on the prediction result, some redundant linking generated by ISESL-SQL may be difficult to understand, such as the linking between token *concerts* and column *concert id* of the exact match based linking result in Figure 1(a). To alleviate this problem, the graph regularization module leverages the schema mention information from SQL as implicit supervision to help regulate the schema information. To summarize, the main contributions of this work are as follows:

- We propose a probing method to extract schema linking graphs from PLMs in an unsupervised manner, which makes the Text-to-SQL model more robust under challenging settings like synonym substitution.
- To the best of our knowledge, we are the first to introduce the graph structure learning methods into schema linking and Text-to-SQL, which refines the initial schema linking graph iteratively during model training.
- We design a graph regularization loss to optimize the schema information in the schema-linking graph.
- We show that ISESL-SQL outperforms strong baselines across three benchmarks (Spider, Spider-SYN and Spider-DK). [1]

2 METHODS

The main framework of our proposed ISESL-SQL is illustrated in Figure 2, we first probe the initial schema linking graph from PLMs (Section 2.2). Then during the training process, the schema linking graph is iteratively refined (Section 2.3). A RAGT graph encoder is adopted to encode the schema linking graph(Section 2.4) and the final SQL statement is generated by a Text-to-SQL Decoder module (Section 2.5). A graph regularization module is also applied in the training process (Section 2.6).

2.1 Problem Definition

We first formulate the Text-to-SQL task as follows. Given a natural language question $Q = (q_1, q_2, \cdots, q_{|Q|})$ and database schema S, which contains multiple tables $T = \{t_1, t_2, \cdots, t_{|T|}\}$ and multi columns $C = \{c_1, \ldots, c_{|C|}\}$, the goal of Text-to-SQL system is to generate a SQL query y.

As a graph encoder is included in our framework, we give a definition to the question-schema graph. The entire graph can be represented as $G = (\mathrm{V}, E)$, which contains all three types of mentioned nodes above. The node sets $V = Q \cup T \cup C$, in which the number of nodes $|V| = |Q| + |T| + |C|$. Here, we denote $S = T \cup C$ to represent all the schema nodes. The edge set $E = E_Q \cup E_S \cup E_{Q \leftrightarrow S}$ contains three kinds of edges: E_Q, E_S, and $E_{Q \leftrightarrow S}$. E_Q contains the edges inside questions tokens, which are defined by the sequence adjacency relationship, E_S includes the edges inside schema items, which are defined by the pre-existing database relations (e.g., primary key relation). Meanwhile, $E_{Q \leftrightarrow S}$ contains the edges between question nodes and schema nodes, which are generated by the schema linking component in Text-to-SQL models. Our work mainly focuses on the construction of $E_{Q \leftrightarrow S}$. The total relation matrix of the graph can be defined as follows:

$$\mathbf{E} = \begin{bmatrix} \mathbf{E}_Q^{|Q| \times |Q|} & \mathbf{E}_{Q \leftrightarrow S} \\ \mathbf{E}_{Q \leftrightarrow S}^T & \mathbf{E}_S^{|S| \times |S|} \end{bmatrix}, \tag{1}$$

where $|S| = |C| + |T|$ is the length of the schema items. Note that the graph is heterogeneous and the value in E represents the edge type (e.g., 1 for semantic linking relation, 2 for primary key relation).

[1] Code and data are available at https://github.com/THU-BPM/ISESL-SQL

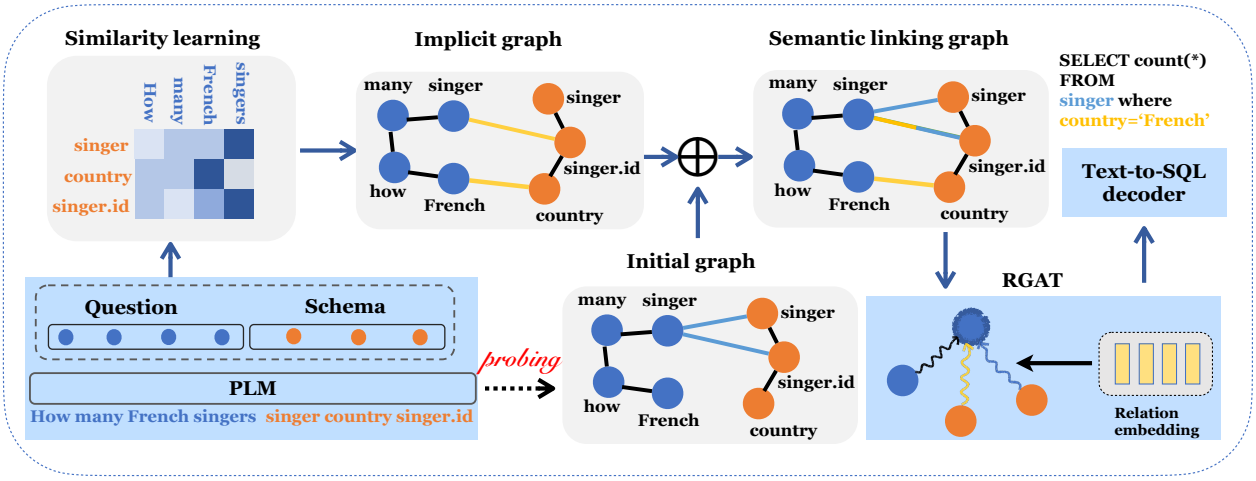

Figure 2: The overall model architecture. We first utilize an unsupervised probing method to construct initial schema-linking graph from PLM (black dashed line). During model training process (blue line), we iteratively learn an adaptive graph structure by parameterized similarity learning. We finally combine these two kinds of graphs and put it into a RGAT network.

2.2 Initial Graph Probing

The initial Graph Probing module aims to obtain a semantic schema linking graph, which is achieved by masking one question token per time and observing how the PLM (e.g., BERT [10]) embeddings of the schema items are affected. Different from the exact matching based method, the generated graph is not dependent on the surface form of the question tokens. Specifically, in this section, we aim to generate an adjacency matrix for an edge type r^{sem} in heterogeneous graph $E_{Q\leftrightarrow S}$ (Eq. 1).

We first concatenate the question tokens and schema items as X:

$$X = \left(q_1, \cdots, q_{|Q|}, s_1, \cdots, s_{|T|+|C|}\right). \tag{2}$$

The PLM maps the input X into hidden representations. The question and schema embeddings can be denoted as $(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{|Q|})$ and $(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{|T|+|C|})$ respectively.

Our goal is to induce a function $f\left(q_i, s_j\right)$ to capture the impact a question word q_i has on the representation of schema item s_j. Inspired by the previous probing method [41], we utilize a two-stage approach to achieve this goal in an unsurpervised manner. First, we replace a question token q_i with spacial token [MASK] and feed the new sequence $X \setminus \{q_i\}$ into the PLM again. Then, the representation of schema item s_j is changed from $\mathbf{s}_j$ to $\mathbf{s}_{j\setminus q_i}$, which is affected by the [MASK] of q_i. We define the $f\left(q_i, s_j\right)$ by measuring the distance between $\mathbf{s}_j$ and $\mathbf{s}_{j\setminus q_i}$ as follows:

$$f\left(q_i, s_j\right) = d\left(\mathbf{s}_{j\setminus q_i}, \mathbf{s}_j\right), \tag{3}$$

where $d(\mathbf{x}, \mathbf{y})$ is the Euclidean distance between two vectors.

To obtain a sparse initial schema linking matrix and avoid the noisy edges with low confidence level, we introduce a pre-defined threshold τ to filter the probing result by:

$$\mathbf{A}_{ij}^{init} = \begin{cases} 0 & \text{if } f\left(q_i, s_j\right) < \tau \\ f\left(q_i, s_j\right) & \text{if } f\left(q_i, s_j\right) \geq \tau \end{cases}, \tag{4}$$

where $\mathbf{A}^{(init)} \in \mathbb{R}^{|Q|\times(|T|+|C|)}$.

2.3 Implicit Graph Learning

To repair the noisy and incomplete initial schema linking graph obtained before training, the Implicit Graph Learning module aims to learn an adaptive graph structure and iteratively refine it during model training.

The goal of the implicit graph learning module is to learn an implicit matrix $\mathbf{A}^{(t)} \in \mathbb{R}^{|Q|\times(|T|+|C|)}$ between question tokens and schema items, where t is the epoch number in training. Specifically, we first calculate the similarity matrix $\mathbf{A}^{(t)}$ by a cosine similarity based function as follows:

$$\mathbf{A}_{ij}^{(t)} = ReLU\left(\frac{< \mathbf{q}_i \times \mathbf{W}_1, \mathbf{s}_j \times \mathbf{W}_2 >}{|\mathbf{q}_i \times \mathbf{W}_1||\mathbf{s}_j \times \mathbf{W}_2|}\right), \tag{5}$$

where $ReLU(x)$ is the rectified linear unit activation function [16] adopted to retain the positive part of its argument. $\mathbf{W}_1$ and $\mathbf{W}_2$ are two learnable weight vectors that transform the question/schema embeddings for similarity calculation and $< \mathbf{x}, \mathbf{y} >$ denotes vector inner product operation.

We could get an weighted similarity matrix from the function in Eq.5. In practice, most schema items only link to one question node, so we propose a graph sparsification component which only keeps the max similarity score for each schema. Our graph sparsification component is defined as follows:

$$\mathbf{A}_{ij}^{t} = \begin{cases} \mathbf{A}_{ij}^{t}, & \mathbf{A}_{ij}^{t} = \text{Max}\left(\mathbf{A}_{j}^{t}\right) \\ 0, & \mathbf{A}_{ij}^{t} \neq \text{Max}\left(\mathbf{A}_{j}^{t}\right) \end{cases}, \tag{6}$$

where $A_j^t = \left[A_{0,j}^t, \ldots, A_{|Q-1|,j}^t\right]$ is the similarity vector of each schema based in Eq.5.

2.4 Graph Encoder

Given the initial schema semantic linking matrix $\mathbf{A}^{init}$ and the implicit semantic schema-linking matrix $\mathbf{A}^{(t)}$, the graph encoder module tries to learn a better question token and schema item

embedding with the enhanced schema linking graph, which could help Text-to-SQL decoder to identify the correct schema item.

The schema linking graph construction procedure at epoch t can be represented as:

$$\widetilde{\mathbf{A}}^{(t)} = \lambda \mathbf{A}^{init} + (1-\lambda)\mathbf{A}^{(t)} \tag{7}$$

where λ is a hyperparameter used to adjust the weights of the two graphs. As shown in Eq.1, the input graph of the graph network is a heterogeneous network which contains different edge types (e.g., semantic linking relation between the question and schema and primary key relation inside schema items). $\widetilde{\mathbf{A}}^{(t)}$ can be seen as a weighed matrix of a specific relation r^{sem}. Given $\widetilde{\mathbf{A}}^{(t)}$, we update the graph $E_{Q\leftrightarrow S}$ in Eq.1 as:

$$[\mathbf{E}_{Q\leftrightarrow S}]_{ij}^{(t)} = \begin{cases} r^{sem}, & \widetilde{\mathbf{A}}^{(t)} > 0 \\ r^{none}, & \widetilde{\mathbf{A}}^{(t)} = 0 \end{cases} \tag{8}$$

In epoch t, the whole graph $\mathbf{E}^{(t)}$ is then generated by replacing $\mathbf{E}_{Q\leftrightarrow S}$ in Eq.1 by $\mathbf{E}_{Q\leftrightarrow S}^{(t)}$. Then we expand the weighted matrix $\widetilde{\mathbf{A}}^{(t)}$ to the whole graph, which can be represented as:

$$\mathbf{M^{(t)}} = \begin{bmatrix} 1^{|Q|\times|Q|} & \widetilde{\mathbf{A}}^{(t)} \\ \widetilde{\mathbf{A}}^{(t)^T} & 1^{|S|\times|S|} \end{bmatrix} \tag{9}$$

As shown in Eq.9, we only calculate the weight matrix between question and schema. The weight of other graphs can be directly set to 1 because there is no uncertainty in these edges.

To process the input heterogeneous graph E, we introduce a relational graph attention network (RGAT) [39] as our graph encoder. The RGAT network utilizes a learnable relational embedding to control the different impacts among edge types. To simplify, we use $\mathbf{X}^l \in \mathbb{R}^{(|Q|+|C|+|T|)\times d}$, to denote the entire node embedding matrix in layer l where d is the GNN embedding size. Similar to previous works [4, 38], we utilize the multi-head scaled dot-product for attention weights calculation. To optimize the weighted matrix $M^{(t)}$ in each epoch, different from previous works, we introduce $M^{(t)}$ to the graph attention calculation process.

Specifically, given the current node representations $\mathbf{X}^l$, graph $E^{(t)}$ and weighted matrix $M^{(t)}$, the attention weight α_{ji}^h is calculated by following equations:

$$\hat{\alpha}_{ji}^h = \left(\mathbf{x}_i^l \mathbf{W}_q^h\right)\left(\mathbf{x}_j^l \mathbf{W}_k^h + \mathbf{M}_{ji}^{(t)}\left[\psi\left(E_{ji}^{(t)}\right)\right]_h^H\right)^{\mathrm{T}}, \tag{10}$$

$$\alpha_{ji}^h = \mathrm{softmax}_j\left(\hat{\alpha}_{ji}^h / \sqrt{d/H}\right), \tag{11}$$

where matrices $\mathbf{W}_q^h, \mathbf{W}_k^h, \mathbf{W}_v^h \in \mathbb{R}^{d\times d/H}$ are trainable transformations and H is the number of heads. Function ψ () is a learnable matrix which transforms the relation type $E_{ji}^{(t)}$ into a d-dim feature vector. $\mathbf{M}_{ji}^{(t)}$ is used to control the influence of relation embedding by a simple multiplication operation. Operator $[\cdot]_h^H$ first evenly splits the vector into H parts and returns the h-th partition.

Then the output representation of current layer $\mathbf{x}_i^{l+1}$ is generated by calculating the relation embedding in a similar way:

$$\hat{\mathbf{x}}_i^l = \Big\|_{h=1}^{H} \sum_{j\in\mathbf{E}_i^{(t)}} \alpha_{ji}^h \left(\mathbf{x}_j^l \mathbf{W}_v^h + \mathbf{M}_{ji}^{(t)}\left[\psi\left(E_{ji}^{(t)}\right)\right]_h^H\right) \tag{12}$$

$$\mathbf{x}_i^{l+1} = FFN\left(\mathrm{LayerNorm}\left(\mathbf{x}_i^l + \hat{\mathbf{x}}_i^l \mathbf{W}_o\right)\right) \tag{13}$$

where matrices $\mathbf{W}_o \in \mathbb{R}^{d\times d}$ is the output transformation, $\|$ represents vector concatenation operation and FFN $(\cdot)$ denotes one feedforward neural network layer.

Since the weighted matrix $M^{(t)}$ is optimized in each epoch, our schema linking graph could be updated iteratively during model training.

The final output of the graph encoder $\mathbf{x^e}$ can be split into question representation $(\mathbf{q}_1^e, \mathbf{q}_2^e, \ldots, \mathbf{q}_{|Q|}^e)$ and schema representation $(\mathbf{s}_1^e, \mathbf{s}_2^e, \ldots, \mathbf{s}_{|T|+|C|}^e)$ for the computation of Text-to-SQL decoder.

2.5 Text-to-SQL Decoder

Given the question representation $(\mathbf{q}_1^e, \mathbf{q}_2^e, \ldots, \mathbf{q}_{|Q|}^e)$ and schema representation $(\mathbf{s}_1^e, \mathbf{s}_2^e, \ldots, \mathbf{s}_{|T|+|C|}^e)$ generated by graph encoder module, Text-to-SQL decoder module aims to generate the corresponding SQL statements.

The architecture of our Text-to-SQL decoder is similar to previous work [44], which generates the abstract syntax tree (AST) of the target SQL y in depth-first traversal order. The action generated by decoder in each timestep is either: (1) An APPLYRULE action that expands the current non-terminal node based on SQL grammar in the partially generated AST. (2) A SELECTTABLE or SELECTCOLUMN action that chooses one table or column item from the encoded schema representation $(\mathbf{s}_1^e, \mathbf{s}_2^e, \ldots, \mathbf{s}_{|T|+|C|}^e)$ [2].

2.6 Graph Regularization

Although the combined graph of the initial graph $\mathbf{A}^{init}$ and implicit graph $\mathbf{A}^{(t)}$ could approach the optimized graph iteratively, the quality of the implicit graph is not guaranteed. Due to that the schema linking module is to find the mentioned schema in question, it is more important for the implicit graph to find the correct schema items than question tokens. In the following, we introduce the auxiliary graph regularization loss to further optimize the schema linking graph

For each SQL query y, we denote the set of column and table names appearing in y as S_{SQL}, which consists of columns and table mentions, i.e., $S_{SQL} = S_{COL} \cup S_{TBL}$. During the training process, we can directly utilize S_{SQL} as the supervision of our graph regularization loss. Specifically, we make the sum of $\mathbf{A}^{(t)}$ in the question dimension approximate the S_{SQL} by binary cross-entropy loss.

$$\mathcal{L}_g = -\sum_{j\in S_{SQL}} log(\sum_i A_{ij}^{(t)}) \tag{14}$$

Note that in Eq.6, only the largest value item of question dimension in $\mathbf{A}^{(t)}$ is retrained. Therefore, the graph regularization loss actually only optimizes the most relevant question of mentioned schema.

This graph regularization auxiliary loss is combined with the main Text-to-SQL loss as follows:

$$\mathcal{L} = \mathcal{L}_{\mathrm{SQL}} + \mu\mathcal{L}_g \tag{15}$$

where μ is a hyperparameter that balances the absolute value of the two losses.

[2] More implementation details are provided in appendix

3 EXPERIMENTS

In this section, we conduct extensive experiments on three different benchmarks to evaluate the effectiveness of our proposed ISESL-SQL model and give detailed analyses to show its advantage.

3.1 Experiment Setup

3.1.1 Datasets Description. We conduct extensive experiments on three public benchmark datasets as follows: (1) **Spider** [46] is a large-scale cross-domain Text-to-SQL benchmark. It contains 8659 training samples across 146 databases and 1034 evaluation samples across 20 databases. We report the exact set match accuracy on the development set, as the test set is not publicly available. (2) **Spider-DK** [14] is a human-curated dataset based on Spider, which samples 535 question-SQL pairs across 10 databases from Spider development set and modifies them to incorporate the domain knowledge. The schema information is implicitly expressed in Spider-DK and thus complex reasoning is required. We train our model on spider training set and test on the Spider-DK development set. (3) **Spider-SYN** [13] is another challenging variant of the Spider dataset. Spider-SYN is constructed by manually modifying natural language questions with synonym substitution, which is more adaptable for the scenario where users do not know the exact schema words mentioned. There are total 1034 samples across 20 databases in Spider-SYN. Our model is trained on the spider training dataset and tested on the Spider-SYN development set.

3.1.2 Baselines. We compare our proposed model with several competing baselines. (1) **LGESQL** [4] is a graph attention network based sequence-to-sequence model with relational GAT and the line graph, which is the previous state-of-the-art Text-to-SQL model. (2) **RATSQL** [38] is a sequence-to-sequence model enhanced by a relational-aware transformer. (3) **ETA** [30] also aims to fix the exact match based schema linking with the pseudo labels generated by PLMs. (4) **SmBoP** [33] introduces a semi-autoregressive bottom-up decoder to generate SQL statements. (5) **DT-Fixup SQL-SP** [43] proposes a theoretically justified optimization strategy to train Text-to-SQL model. (6) **IRNET** [17] proposes an intermediate representation SemSQL to close the gap between natural language and SQL statements. (7) **EditSQL** [47] views SQL as sequences and reuses previous generation results at the token level. (8) **RYANSQL** [7] generates nested queries by recursively yielding its component SELECT statements and uses a sketch-based slot filling approach to predict each SELECT statement. (9) **Global-GNN** [3] proposes a semantic parser that globally reasons about the structure of the output query to make a more contextually informed selection of database constants. Many previous works design adaptive PLM models for specific Text-to-SQL models to achieve better result, such as GAP [35], GRAPPA [45], STRUG [9]. For fair comparison, except for comparing the result on a unified pre-training model BERT-large, we also report the result with model adaptive PLM.

3.1.3 Hyper-parameters. The threshold τ in the initial graph probing process is set to 0.7. In the encoder part, the hidden size d of the RGAT is 256 for GloVe and 512 for all PLMs. The number of RAGT layers L is 8. The number of heads in RGAT's multi-head attention is 8 and the dropout rate of features is set to 0.2. λ is set to 0.2 when combining the initial graph and implicit graph. In the decoder part, following the previous work [4], the dimension of hidden state, action embedding, and node type embedding size is set to 512, 128, and 128 respectively. And the dropout rate for decoder LSTM is 0.2. We use AdamW [32] with learning rate 5e-4 for GloVe and 1e-4 for PLMs. The balancing hyperparameter μ is set to 1 in Eq.15. Furthermore, we use a linear warmup scheduler with a warmup ratio of 0.1. The batch size is set to 20 and the total training epoch is 200.

Model	Spider	Spider-DK	Spider-SYN
Without PLM: GloVe			
Global-GNN + GloVe [3]	52.7	26.0	23.6
EditSQL + GloVe[47]	36.4	31.4	25.3
IRNet + GloVe[17]	53.2	33.1	28.4
RATSQL + GloVe [38]	62.7	35.8	33.6
LGESQL + GloVe [4]	67.6	39.2	40.5
ISESL-SQL + GloVe	68.3 (0.7↑)	42.1 (2.9↑)	44.4 (3.9↑)
With PLM: BERT			
EditSQL + BERT-large [47]	57.6	36.2	34.6
IRNet + BERT-large[17]	53.2	38.6	36.7
RYANSQL + BERT-large [7]	66.6	40.1	47.8
RATSQL + BERT-large [38]	69.7	40.9	48.2
ETA + BERT-large [30]	70.8	41.8	50.6
LGESQL + BERT-large [4]	74.1	44.7	55.1
ISESL-SQL + BERT-large	74.7 (0.6 ↑)	46.2 (1.5↑)	56.8 (1.7↑)
With Model Adaptive PLM			
RATSQL + STRUG [9]	72.6	39.4	48.9
RATSQL + GRAPPA [45]	73.4	38.5	49.1
SmBoP + GRAPPA [33]	74.7	42.2	48.6
RATSQL + GAP [35]	71.8	44.1	49.8
DT-Fixup SQL-SP + RobERTa [43]	75.0	40.5	50.4
LGESQL + ELECTRA-large [4]	75.1	48.4	62.6
ISESL-SQL + ELECTRA-large	**75.8** (0.7↑)	**50.0** (1.6↑)	**64.8** (2.2↑)

Table 1: Exact match accuracy (%) on three different benchmarks: Spider,Spider-DK and Spider-SYN.

3.2 Main Results

Table 1 shows the exact match accuracy on three benchmarks with the exact match average accuracy of 3 runs. LGESQL is the previous state-of-the-art model in all three embedding configurations (without PLM, with BERT-large and with model adaptive PLM). In general, ISESL-SQL could outperform previous baselines in all configurations. More specifically, with GloVe word vectors, our model could surpass the previous best model by an average of 2.5% over all benchmarks. Similarly, ISESL-SQL could achieve an average performance boost of 1.3% with a unified pre-training model BERT-large over the three benchmarks. Furthermore, our ISESL-SQL achieves state-of-the-art performance with ELECTRA-large [8] on the model adaptive PLM setting.

We observe that ISESL-SQL could obtain greater improvement on Spider-SYN and Spider-DK benchmarks than standard spider benchmark, which proves the robustness of ISESL-SQL in challenging datasets. Also, since the ability for GloVe word vectors to capture semantic linking is inferior to PLM embeddings, ISESL-SQL achieves more performance boost with GloVe word vectors setting.

3.3 Ablation Studies

We conduct ablation studies to show the effectiveness of different modules of ISESL-SQL to the overall improved performance.

Technique	Spider	Spider-SYN	Spider-DK
ISESL-SQL	**75.8**	**64.8**	**50.0**
w/o initial graph probing	75.0 (0.8↓)	62.8 (2.0↓)	48.5(1.5↓)
w/o implicit graph learning	74.6 (1.2↓)	63.3 (1.5↓)	48.7(1.3↓)
w/o schema linking	73.4 (2.5↓)	63.8 (1.0↓)	49.3 (0.7↓)
w/o graph regularization	74.8 (1.0↓)	63.9 (0.9↓)	49.2 (0.8↓)
w exact match schema linking	73.6 (2.2↓)	62.3 (2.5↓)	47.6 (2.4↓)

Table 2: Ablation study of different modules.

Model	Col_P	Col_R	Col_F	Tab_P	Tab_R	Tab_F
N-gram Match	61.4	69.4	65.1	78.2	69.6	73.6
SIM	16.6	8.0	10.8	8.5	11.6	9.8
CONTRAST	83.7	68.4	75.3	84.0	76.9	80.3
ETA	86.1	79.3	82.5	81.1	85.3	83.1
$\text{SLSQL}_L^{\heartsuit}$	82.6	82.0	82.3	80.6	84.0	82.2
ISESL-SQL	**87.2**(1.1↑)	**85.3**(3.3↑)	**86.2**(3.7↑)	**89.4**(5.4↑)	**87.1**(1.8↑)	**88.2**(5.1↑)

Table 3: Schema linking experimental results on spider dev sets. $^{\heartsuit}$ means the model uses schema linking supervision.

ISESL-SQL w/o initial graph pruning is the proposed model without the initial probed graph and only keeps the learned graph during training. ISESL-SQL w/o implicit graph learning only adopts the initial probed graph with no more graph optimization process. ISESL-SQL w/o schema linking replaces all the edges in $\mathbf{E}_{Q\leftrightarrow S}$ (Eq.1) to none-relation edge type r^{none}. ISESL-SQL w/o graph regularization removes the graph regularization loss and only keeps the Text-to-SQL loss during training. ISESL-SQL w exact match schema linking replaces our semantic schema linking edge with the the exact match schema linking edges.

A general conclusion from ablation results in Table 2 is that all modules contribute positively to the improved performance. More specifically, ISESL-SQL w/o initial graph probing gives us 1.4% less performance averaged on all datasets. Similarly, implicit graph learning gives 1.3% performance boost in average over all benchmarks. Removing the graph regularization loss leads to an average of 0.9% performance drop. Furthermore, when totally removing the schema linking edges, ISESL-SQL w/o schema linking brings an average performance drop of 1.4%. Compared with the exact match schema linking method, our ISESL-SQL gives an average improvement of 2.4%.

We can discover that the initial graph probing module contributes more in spider-SYN and spider-DK than spider benchmark, which proves the importance of semantic linking edges in challenging settings. Also, the exact match schema linking method is even worse than no schema linking settings, which shows it vulnerability in challenging settings.

3.4 Schema Linking Analysis

To better demonstrate the quality of our constructed schema linking graph, we compare the schema information (tables and columns) produced by ISESL-SQL with human annotations.

Following the previous works [25, 30], we report the micro average precision, recall and F1-score for both columns (Col_P , Col_R, Col_F) and tables (Tab_P ,Tab_R, Tab_F). The metric focus on whether the correct schema item is identified.

We consider five strong baselines for comparison. (1) **N-gram Matching** links all n-gram ($n \leq 5$) phrases in a natural language question to schema items by fuzzy string matching. (2) **SIM** computes the cosine similarity between question tokens and schema items using PLM embeddings without task specific fine-tuning. (3) **CONTRAST** learns by comparing the aggregated embedding scores of mentioned schemas with unmentioned ones in a contrastive learning style, as done in Liu et al. [29]. (4) $\textbf{SLSQL}_L$ [25] is trained with full schema linking supervision by a learnable schema linking module. (5) **ETA** [30] trains the schema linking module using pseudo alignment as supervision generated from PLMs.

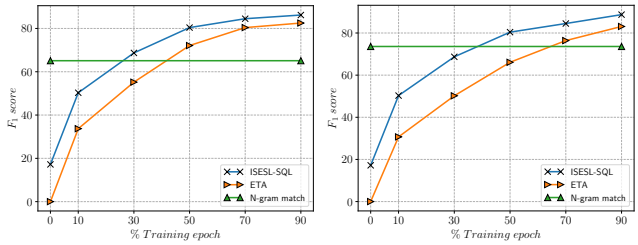

Figure 3: F1 results of the column match (left) and table match (right) accuracy during the training process on spider development set.

Table 3 shows the experimental results on the schema linking accuracy. Our proposed ISESL-SQL model could surpass all weakly supervised and unsupervised methods by a large margin. Specifically, our method brings an improvement of 3.7% on Col_F and 5.1% on Tab_F over the previous best baseline ETA [30]. SIM is a similar method to our initial graph probing component which constructs schema linking graph in an unsupervised way. We discover that our method outperforms SIM by a huge margin, which indicates the effectiveness of our iterative graph learning process. Surprisingly, our ISESL-SQL method could outperform SLSQL_L which is trained in a fully supervised manner. This indicates that the weak supervision from schema information could achieve a similar result with fully supervision.

To further investigate how our model learns the schema linking graph iteratively during the training process, we visualize the changing trend of schema match F1 score during the training process in the spider development set. As shown in Figure 3, our method could outperform the previous best baseline ETA during the entire training process. And after epoch 45, our ISESL-SQL surpasses the N-gram match based method. Our model could achieve the best schema linking result before epoch 100 in both column matching and table matching. Also, it can be seen that our ISESL-SQL model has an initial F1 score before training, which is the result of the initial graph probing. These results prove our model could iteratively refine the schema linking graph during the training process.

3.5 Component Matching Analysis

To better analyze the reasons for the performance improvements achieved by our ISESL-SQL, we perform a detailed analysis of the matching accuracy of different components of the SQL statements on the Spider-SYN benchmark. As shown in Table 4, the performance improvement mainly comes from the components including schema items, such as *SELECT*, *SELECT (no AGG)*, *WHERE* and *WHERE (op)*. Specifically, on the *WHERE (op)* component, our

SQL component	RATSQL	LGESQL	ISESL-SQL
SELECT	74.6	84.0	84.7 (0.7 ↑)
SELECT (no AGG)	76.3	85.2	86.3 (1.1 ↑)
WHERE	73.1	71.8	74.7 (1.6 ↑)
WHERE(no OP)	77.3	76.2	79.5 (2.2 ↑)
GROUP BY (no HAVING)	67.0	81.6	81.0 (0.6 ↓)
GROUP BY	63.1	79.4	78.3 (1.1 ↓)
ORDER BY	79.2	82.9	83.0 (0.1 ↑)
AND/OR	98.3	97.7	98.1 (0.2 ↓)
IUE	27.5	51.2	50.3 (0.9 ↓)
KEYWORDS	87.4	87.3	88.5 (1.1 ↑)

Table 4: F1 scores of component matching of RATSQL, LGESQL and our model on Spider-SYN benchmark.

Model	Spider	Spider-SYN
ISESL-SQL	75.8	64.8
ISESL-SQL + Oracle columns	80.8 (5.0↑)	78.2(13.4↑)
ISESL-SQL + Oracle tables	79.1 (3.3↑)	76.9(12.1↑)
ISESL-SQL + Oracle schema	83.2 (7.4↑)	82.1 (17.3↑)
ISESL-SQL + Oracle schema linking	**83.5** (7.7↑)	**82.7**(17.9↑)

Table 5: Exact match accuracy (%) on Spider and Spider-Syn benchmarks when oracle schema information is provided.

ISESL-SQL model outperforms the previous best baseline by 2.2%. On other components where schema items are not included, the performance remains unchanged or slightly drops on some components, such as *IEU* and *AND/OR*. The observation shows a high-quality schema linking graph could help the Text-to-SQL model find the correct schema items.

3.6 Oracle Information Provided Analysis

One natural question is how much improvement will be made when the column and table mentioned information (oracle schema information) is provided, As shown in Table 5, we report the exact match accuracy of our ISESL-SQL model on Spider and Spider-SYN benchmarks with oracle schema information provided. Specifically, we directly change the implicit graph matrix $\widetilde{\mathbf{A}}^{(t)}$ in Eq. 7 based on the oracle information.

From Table 5, we could discover that: 1) Oracle schema linking information could bring a huge improvement (7.7% and 17.9% in Spider and Spider-SYN benchmarks respectively) to the final Text-to-SQL accuracy, which proves the importance of schema linking component. 2) When oracle schema linking information is provided, the results on the Spider-SYN benchmark (80.1%) will be close to those on the Spider(83.1%) , which demonstrates that the original performance drops in the Spider-SYN benchmark mainly come from the corrupted schema linking graph. 3) Only providing oracle schema information (which schema item appears) could achieve a similar improvement compared to the oracle schema linking, which could demonstrate our assumption that schema information is more important in the schema linking graph. 4) Oracle column information contributes more than the oracle table information, which is because the column information is more difficult to capture.

Based on the above results, we can conclude that there is a huge potential for improvement by designing a better schema linking component, which could be an important direction for future work.

Question: Find the distinct breed type and size type combinations for **puppies.**

LGESQL: `SELECT breed_name, size_code FROM Breeds`

ISESL-SQL: `SELECT breed_name, size_code FROM Dogs`

Question: Please show the **record type** of **ensembles** in ascending order of count.

LGESQL: `SELECT performance.Type FROM performance GROUP BY performance.Type ORDER BY COUNT(*) ASC`

ISESL-SQL: `SELECT Major_Record_Format FROM orchestra GROUP BY Major_Record_Format ORDER BY COUNT(*) ASC`

Question: What is the average and highest **capacities** for all stations?

LGESQL: `SELECT AVG(stadium.Capacity), MAX(stadium.Highest) FROM stadium`

ISESL-SQL: `SELECT avg(capacity) , max(capacity) FROM stadium`

Question: Of all the competitors who got voted, what is the competitor number and name of the competitor who got **least votes** ?

ISESL-SQL: `SELECT contestant_name FROM contestants JOIN votes ORDER BY votes.vote_id LIMIT 1`

Gold: `SELECT contestant_name FROM contestant JOIN votes GROUP BY contestant_number ORDER BY count(*) LIMIT 1`

Table 6: Case study: the first two cases are sampled from Spider-SYN and the last two cases are sampled from Spider-DK. The first three cases are positive cases while the last one is a negative case.

3.7 Case study

To intuitively show the effectiveness of our model, we select four cases from Spider-SYN and Spider-DK benchmarks to compare the generated SQL statements of our ISESL-SQL model and LGESQL. The first two cases are from the Spider-SYN benchmark and the last two cases are from the Spider-DK benchmark. As shown in Table 6 we can observe that our model could generate correct SQL even in synonym substitution scenario. As in the first case, when the schema-related token *dogs* is replaced with *puppies* in the question, LGESQL fails to identify table name *dogs* while our ISESL-SQL model could successfully recognize the correct table. In the third case, the token *highest* in question could be matched to the column *highest* via the exact match based method, which leads to the error in LGESQL. Since our method uses a semantic-based approach to do matching, the above problem could be avoided in most cases. However in the more complicated cases (e.g. the forth case in Table 6), ISESL-SQL may fail to recognize the complex structure information.

Figure 4 shows the alignment matrix between question and schema during the training process. We take the sentence *Show the ages for all French Musicians* and the schema in the database *concert_singer* as an example. The first subfigure shows the alignment matrix generated by the initial graph probing component before training. Although the linking of column 'age' and table 'singer' is identified, the initial graph probing component still fails to capture the linking between *French* and country. As we can see, during the training process, the graph is iteratively optimized. The correct linking is emphasized in epoch 50. Finally, in epoch 100, our method could almost exclusively focus on the correct linking. The whole process demonstrates that our ISESL-SQL method could generate better schema linking information iteratively during training.

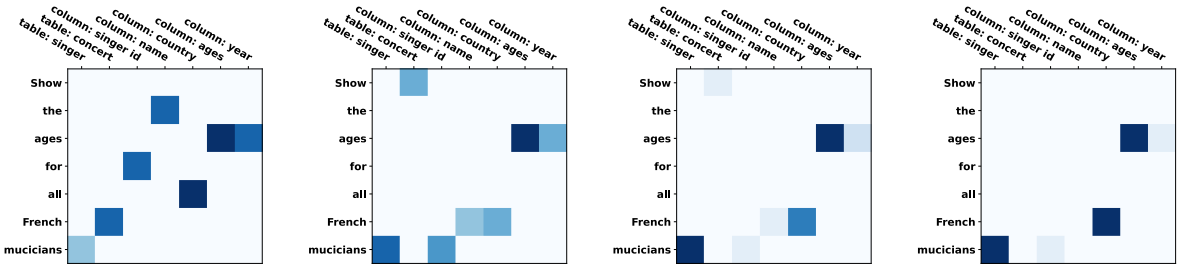

Figure 4: Alignment matrix between the question "show the ages for all French musicians" and the database 'concert_singer' schema in training epoch 0, 25, 50, and 100.

4 RELATED WORK

4.1 Schema Linking in Text-to-SQL Parsing

Text-to-SQL Parsing is an essential sub-field of Natural Language Processing and could benefit many other tasks such as Question Answering and Information Extraction [5, 18–20, 27, 28, 31]. Schema linking is an indispensable module in recent Text-to-SQL models, which establishes a link between question tokens and schema items [15]. Many previous works treat schema linking as a minor preprocessing procedure [4, 17, 26, 38, 42], which use surface form match based method to find the occurrences of the schema names in the question. However, these methods may fail in synonym and typo scenarios. Other works implement schema-linking by learning a similarity score between a word and a schema item [2, 24], but still suffer from the limitation of the static word embedding. Guo et al. [17] and Wang et al. [38] conduct an ablation study on schema linking, and the results show that removing the extra schema linking causes the biggest performance decline compared to removing other removable modules. To better investigate the influence of schema linking, Lei et al. [25] and Taniguchi et al. [36] invest human resources to annotate schema linking information in dataset and employ the full supervision to train Text-to-SQL models. The result shows that with gold schema linking information as training data, current Text-to-SQL models can exceed the baseline by a very large gap. Dong et al. [11] utilize implicit linking supervision to train their schema linking model with reinforcement learning method. Liu et al. [30] train the schema linking module using pseudo alignment as supervision from PLMs, but the pseudo alignment could still be noisy. Our `ISESL-SQL` iteratively refines the schema linking graph using both supervision from SQL generation task and the schema mention information in SQL statements.

4.2 Graph Learning based Models

Graph neural networks (GNNs) are neural models that capture the dependence of graphs[48]. The structure of the graph is normally labeled by experts [34] , pre-processed with existing relation parsing tools [40] or precomputed by the k-nearest neighbor algorithm [1]. However, the graph structure generated by these methods may contain missing or noisy edges which may not be optimal for downstream learning tasks. To alleviate this problem, some works propose a robust graph learning schema by removing noise and errors in the raw data adaptively [22]. Similar to these works, graph attention network is proposed to use self-attention mechanism to reweight edge importance [37]. As these robust learning approaches still cannot handle the missing edges, adaptive graph structure learning methods are proposed to facilitate downstream graph-based tasks [21]. These adaptive graph construction approaches typically utilize a parameterized graph similarity metric learning function to learn an adjacency matrix by considering pair-wise node similarity in the embedding space [40]. These models only perform graph structure learning for one time which is not enough. To better optimize graph structure and downstream tasks jointly, Chen et al. [6] proposed an iterative deep graph learning model to let graph learning models and downstream tasks optimize together. In the scenarios when the input graph is not available, LDS [12] is proposed to learn the graph structure by solving a bilevel program that learns the discrete probability over the graph egdes. However, these models cannot be applied to Text-to-SQL directly because the graph in Text-to-SQL model is heterogeneous which contains different types of nodes and edges. In this paper, we propose a framework to introduce the graph structure learning network into a modified version of Relation-aware graph attention network [39] to enhance Text-to-SQL models.

5 CONCLUSION

In this paper, we propose a framework named `ISESL-SQL` to build a semantic enhanced schema-linking graph for the Text-to-SQL task. Different from the previous exact match-based schema linking method, the schema linking graph generated by our method is accurate and robust in various settings such as synonym substitution. We perform extensive experiments on three benchmarks and the results demonstrate the effectiveness of our method.

Here, we list several main findings as follows. 1) The schema linking graph generated by PLM could help the Text-to-SQL model find the correct schema information. 2) Implicit graph learning module plays a very important role to capture the correct schema linking information. 3) In the schema linking graph, the correctness of the schema items is much more important than the question tokens, which indicates that weak supervision could work in most scenarios. 4) When oracle schema information is provided, the Text-to-SQL accuracy will be greatly improved, which implies a huge potential for improvement. 5) The auxiliary task graph regularization could consistently yield improvements on multiple benchmarks.

ACKNOWLEDGMENTS

The work was supported by the National Key Research and Development Program of China (No. 2019YFB1704003), the National Nature Science Foundation of China (No. 62021002 and No. 71690231), Tsinghua BNRist and Beijing Key Laboratory of Industrial Big Data System and Application.

REFERENCES

[1] David C. Anastasiu and George Karypis. 2015. L2Knng: Fast Exact K-Nearest Neighbor Graph Construction with L2-Norm Pruning. In *Proc. of CIKM*. 791–800.
[2] Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. In *Proc. of ACL*. 4560–4565.
[3] Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Global Reasoning over Database Structures for Text-to-SQL Parsing. In *Proc. of EMNLP-IJCNLP*. 3657–3662.
[4] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. In *Proc. of ACL-IJCNLP*. 2541–2555.
[5] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proc. of ACL*. Association for Computational Linguistics, Vancouver, Canada, 1870–1879. https://doi.org/10.18653/v1/P17-1171
[6] Yu Chen, Lingfei Wu, and Mohammed J. Zaki. 2020. Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. In *NeurIPS*.
[7] DongHyun Choi, Myeongcheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2021. RYANSQL: Recursively Applying Sketch-based Slot Fillings for Complex Text-to-SQL in Cross-Domain Databases. *Comput. Linguistics* (2021), 309–332.
[8] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *ICLR*.
[9] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proc. of NAACL-HLT*. 1337–1350.
[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of NAACL-HLT*. 4171–4186.
[11] Zhen Dong, Shizhao Sun, Hongzhi Liu, Jian-Guang Lou, and Dongmei Zhang. 2019. Data-Anonymous Encoding for Text-to-SQL Generation. In *Proc. of EMNLP-IJCNLP*. 5405–5414.
[12] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning discrete structures for graph neural networks. In *Proc. of ICML*. 1972–1982.
[13] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021. Towards Robustness of Text-to-SQL Models against Synonym Substitution. In *Proc. of ACL-IJCNLP*. 2505–2515.
[14] Yujian Gan, Xinyun Chen, and Matthew Purver. 2021. Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization. In *Proc. of EMNLP*. 8926–8931.
[15] Yujian Gan, Matthew Purver, and John R. Woodward. 2020. A Review of Cross-Domain Text-to-SQL Models. In *Proc. of COLING*. 108–115.
[16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proc. of AISTATS*. 315–323.
[17] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *Proc. of ACL*. 4524–4535.
[18] Xuming Hu, Fukun Ma, Chenyao Liu, Chenwei Zhang, Lijie Wen, and Philip S Yu. 2021. Semi-supervised Relation Extraction via Incremental Meta Self-Training. In *Proc. of EMNLP: Findings*.
[19] Xuming Hu, Lijie Wen, Yusong Xu, Chenwei Zhang, and Philip Yu. 2020. SelfORE: Self-supervised Relational Feature Learning for Open Relation Extraction. In *Proc. of EMNLP*. Association for Computational Linguistics, Online, 3673–3682.
[20] Xuming Hu, Chenwei Zhang, Yawen Yang, Xiaohe Li, Li Lin, Lijie Wen, and S Yu Philip. 2021. Gradient Imitation Reinforcement Learning for Low Resource Relation Extraction. In *Proc. of EMNLP*. 2737–2746.
[21] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. 2019. Semi-supervised learning with graph learning-convolutional networks. In *Proc. of AAAI*. 11313–11320.
[22] Zhao Kang, Haiqi Pan, Steven CH Hoi, and Zenglin Xu. 2019. Robust graph learning from noisy data. *IEEE transactions on cybernetics* 50, 5 (2019), 1833–1843.
[23] George Katsogiannis-Meimarakis and Georgia Koutrika. 2021. A Deep Dive into Deep Learning Approaches for Text-to-SQL Systems. In *Proc. of SIGMOD*. 2846–2851.
[24] Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural Semantic Parsing with Type Constraints for Semi-Structured Tables. In *Proc. of EMNLP*. 1516–1526.
[25] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the Role of Schema Linking in Text-to-SQL. In *Proc. of EMNLP*. 6943–6954.
[26] Shu'ang Li, Xuming Hu, Li Lin, Aiwei Liu, Lijie Wen, and Philip S Yu. 2022. A Multi-level Supervised Contrastive Learning Framework for Low-Resource Natural Language Inference. *arXiv preprint arXiv:2205.15550* (2022).
[27] Shu'ang Li, Xuming Hu, Li Lin, and Lijie Wen. 2022. Pair-Level Supervised Contrastive Learning for Natural Language Inference. In *Proc. of ICASSP*.
[28] Li Lin, Yixin Cao, Lifu Huang, Shuang Li, Xuming Hu, Lijie Wen, and Jianmin Wang. 2022. Inferring Commonsense Explanations as Prompts for Future Event Generation. In *Proc. of SIGIR*.
[29] Qian Liu, Yihong Chen, Bei Chen, Jian-Guang Lou, Zixuan Chen, Bin Zhou, and Dongmei Zhang. 2020. You Impress Me: Dialogue Generation via Mutual Persona Perception. In *Proc. of ACL*. 1417–1427.
[30] Qian Liu, Dejian Yang, Jiahui Zhang, Jiaqi Guo, Bin Zhou, and Jian-Guang Lou. 2021. Awakening Latent Grounding from Pretrained Language Models for Semantic Parsing. In *Findings of ACL-IJCNLP*. 1174–1189.
[31] Shuliang Liu, Xuming Hu, Chenwei Zhang, Shu'ang Li, Lijie Wen, and Philip S. Yu. 2022. HiURE: Hierarchical Exemplar Contrastive Learning for Unsupervised Relation Extraction. In *Proc. of NAACL*.
[32] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *ICLR*.
[33] Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *Proc. of NAACL-HLT*. 311–324.
[34] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.
[35] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cícero Nogueira dos Santos, and Bing Xiang. 2021. Learning Contextual Representations for Semantic Parsing with Generation-Augmented Pre-Training. In *Proc. of AAAI*. 13806–13814.
[36] Yasufumi Taniguchi, Hiroki Nakayama, Kubo Takahiro, and Jun Suzuki. 2021. An Investigation Between Schema Linking and Text-to-SQL Performance. *arXiv preprint arXiv:2102.01847* (2021).
[37] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[38] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proc. of ACL*. 7567–7578.
[39] Kai Wang, Weizhou Shen, Yunyi Yang, Xiaojun Quan, and Rui Wang. 2020. Relational Graph Attention Network for Aspect-based Sentiment Analysis. In *Proc. of ACL*. 107736.
[40] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. 2021. Graph Neural Networks for Natural Language Processing: A Survey. *arXiv preprint arXiv:2106.06090* (2021).
[41] Zhiyong Wu, Yun Chen, Ben Kao, and Qun Liu. 2020. Perturbed Masking: Parameter-free Probing for Analyzing and Interpreting BERT. In *Proc. of ACL*. 4166–4176.
[42] Peng Xu, Dhruv Kumar, Wei Yang, Wenjie Zi, Keyi Tang, Chenyang Huang, Jackie Chi Kit Cheung, Simon J.D. Prince, and Yanshuai Cao. 2021. Optimizing Deeper Transformers on Small Datasets. In *Proc. of ACL-IJCNLP*. 2089–2102.
[43] Peng Xu, Dhruv Kumar, Wei Yang, Wenjie Zi, Keyi Tang, Chenyang Huang, Jackie Chi Kit Cheung, Simon J. D. Prince, and Yanshuai Cao. 2021. Optimizing Deeper Transformers on Small Datasets. In *Proc. of ACL-IJCNLP*. 2089–2102.
[44] Pengcheng Yin and Graham Neubig. 2017. A Syntactic Neural Model for General-Purpose Code Generation. In *Proc. of ACL*. 440–450.
[45] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2021. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing. In *ICLR*.
[46] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proc. of EMNLP*. 3911–3921.
[47] Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions. In *Proc. of EMNLP-IJCNLP*. 5338–5349.
[48] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81.

A DECODER ARCHITECTURE

Given the question representation $(\mathbf{q}_1^e, \mathbf{q}_2^e, \ldots, \mathbf{q}_{|Q|}^e)$ and schema representation $(\mathbf{s}_1^e, \mathbf{s}_2^e, \ldots, \mathbf{s}_{|T|+|C|}^e)$ generated by graph encoder module, Text-to-SQL decoder module aims to generate the corresponding SQL statements.

The architecture of our Text-to-SQL decoder is similar to previous work [44], which generates the abstract syntax tree (AST) of the target SQL y in depth-first traversal order. The decoder utilizes a LSTM to generate actions in each timestep which is either: (1) An APPLYRULE action that expands the current non-terminal node based on SQL grammar in the partially generated AST. (2) A SELECTTABLE or SELECTCOLUMN action that chooses one table or column item from the encoded schema representation $(\mathbf{s}_1^e, \mathbf{s}_2^e, \ldots, \mathbf{s}_{|T|+|C|}^e)$.

Formally, the whole process can be defined as

$$P(y \mid \mathbf{X}) = \prod_i P\left(a_i \mid a_{<i}, \mathbf{X}\right) \tag{16}$$

where X is the combined question and schema representation, $a_{<i}$ is all the previous actions before step i. Specifically, in each step, the decoder network updates the state as follows:

$$\mathbf{m}_i, \mathbf{h}_i = LSTM\left(\left[\mathbf{a}_{i-1}; \mathbf{a}_{p_i}; \mathbf{h}_{p_i}; \mathbf{t}_i\right], \mathbf{m}_{i-1}, \mathbf{h}_{i-1}\right) \tag{17}$$

where $\mathbf{m}_i$ and $\mathbf{h}_i$ are the cell state and output of the i-th timestep, $\mathbf{a}_{i-1}$ is previous action embedding, $\mathbf{a}_{p_i}$ is the parent embedding of current node, $\mathbf{h}_{p_i}$ is the parent cell state, $\mathbf{t}_i$ is the type embedding of current node.

To this end, the operation APPLYRULE can be represented as

$$P\left(a_i = \text{ APPLYRULE } [R] \mid a_{<i}, \mathbf{X}\right) = \text{softmax}_R\left(\mathbf{h}_i \mathbf{W}_{\text{R}}\right) \tag{18}$$

where $\mathbf{W}_{\text{R}}$ transforms the LSTM output into action rule logits.

The SELECTTABLE is implemented by calculating the attention between LSTM hidden state $\mathbf{h}_i$ and table representation $\mathbf{x}_{t_j}$:

$$\begin{aligned} P\left(a_i = \text{SELECTTABLE}\left[t_j\right] \mid a_{<i}, \mathbf{X}\right) = \\ \text{softmax}_j\left(\mathbf{h}_i \mathbf{W}_{tq}\right)\left(\mathbf{x}_{t_j} \mathbf{W}_{tk}\right)^{\text{T}} \end{aligned} \tag{19}$$

And the SELECTCOLUMN operation is defined in a similar way.

The loss of Text-to-SQL is defined as follows:

$$\mathcal{L}_{\text{SQL}} = \sum_{i=1}^{T} \log P\left(a_i \mid a_{<i}, \mathbf{X}\right) \tag{20}$$

where T is the total number of actions in the abstract syntax tree.