Re-examining the Role of Schema Linking in Text-to-SQL

Wenqiang Lei[*†] , **Weixin Wang**[*†], **Zhixin Ma**[†], **Tian Gan**[‡], **Wei Lu**[§], **Min-Yen Kan**[†], **Tat-Seng Chua**[†]
†National University of Singapore
‡Shandong University
§Singapore University of Technology and Design
wenqianglei@gmail.com weixin.wang@comp.nus.edu.sg
zhixinma97@gmail.com gantian@sdu.edu.cn
luwei@sutd.edu.sg kanmy@comp.nus.edu.sg dcscts@nus.edu.sg

Abstract

In existing sophisticated text-to-SQL models, schema linking is often considered as a simple, minor component, belying its importance. By providing a schema linking corpus based on the Spider text-to-SQL dataset, we systematically study the role of schema linking. We also build a simple BERT-based baseline, called **S**chema-**L**inking SQL (SLSQL) to perform a data-driven study. We find when schema linking is done well, SLSQL demonstrates good performance on Spider despite its structural simplicity. Many remaining errors are attributable to corpus noise. This suggests schema linking is the crux for the current text-to-SQL task. Our analytic studies provide insights on the characteristics of schema linking for future developments of text-to-SQL tasks.[1]

1 Introduction

Structured Query Language (SQL), while exact and powerful, suffers from a complex grammar presenting significant challenges for laymen to write queries. Automatically parsing natural language into SQL (text-to-SQL) thus has huge potential, as it would enable lay users to mine the world's structured data using natural language queries.

To achieve practical text-to-SQL workflow, a model needs to correlate natural language queries with the given database. Therefore, **schema linking** is considered helpful for text-to-SQL parsing (Guo et al., 2019; Bogin et al., 2019b; Dong et al., 2019; Wang et al., 2020). Here, schema linking means identifying references of columns, tables and condition values in natural language queries. For example, for the question "Find the names of schools that have a donation with amount above 8.5" (shown with relevant tables in Figure 1), "name" is a *column reference* to `school.name`, "donation" a *table reference* to `endowment`, and "8.5" and "sale" are *value references*, corresponding to the condition values in the SQL query.

Existing solutions largely treat *schema linking* as a minor component implemented with simple string matching (Guo et al., 2019; Yu et al., 2018a; Lin et al., 2019) heuristics to support sophisticated text-to-SQL models. An exception is Dong et al. (2019), which framed schema linking as a task to be solved by sequential tagging. While they did show the importance of schema linking, how it contribute to text-to-SQL task performance remains unanswered as there is no annotated corpus to analyze.

To address these shortcomings, we perform an in-depth study on the role of schema linking in text-to-SQL parsing. Intuitively, schema linking helps both *cross-domain generalizability* and *complex SQL generation*, which have been identified as the current bottlenecks of the text-to-SQL task (Finegan-Dollak et al., 2018; Yu et al., 2018c). By *cross-domain generalizability*, we refer to the proper separation of training and testing instances and databases, requiring a model to infer against with arbitrary databases where the schema and the domain are previously unknown. This means the model must be aware of what tables and columns are involved in the question — exactly what schema linking does. Schema linking indirectly addresses the *complex SQL generation* challenge: the writing of SQL queries comprising a mixture of `select`, `group by`, and nested clauses. Generating such queries requires the modeling of complex semantic dependencies in the input and to manage complex SQL grammar during decoding. As discussed in Dong et al. (2019), detecting and removing domain-specific words from the model's purview allows the model to focus on learning syntactic conversion between natural language and SQL, reducing the

[*] Equal contribution.
[1] Our code and annotation are available at https://github.com/WING-NUS/slsql.

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 6943–6954,
November 16–20, 2020.

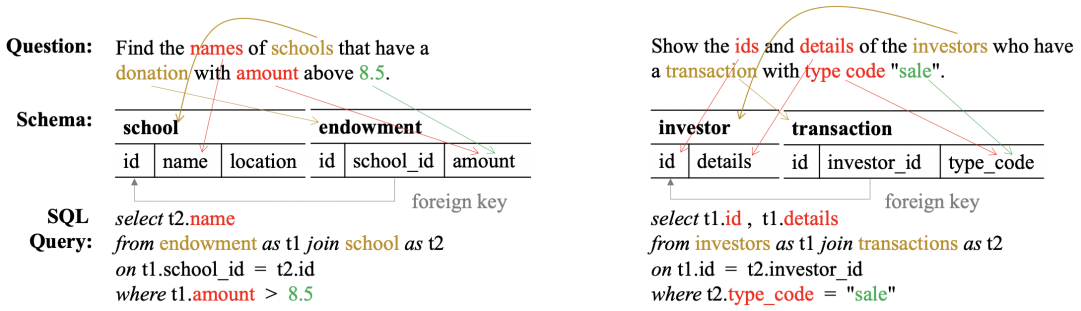

Figure 1: Two examples of schema linking. Column, table and value references are marked in red, yellow and green, respectively. The arrows of column and table references indicate their respective referents in the schema. For value references, the arrows point to the columns they compare with.

input's syntactic sparsity. For example, if we view the words linked with the schema as placeholders, the two natural language queries in Figure 1 can be deemed syntactically similar.

To perform a systematic data-driven study, we annotate and contribute ground truth schema linking data for the publicly-available training and development set of the Spider dataset (Yu et al., 2018c). We then build a simple BERT baseline, named **S**chema-**L**inking SQL (SLSQL), which links the schema in a natural language query and parses the SQL query with awareness of the predicted schema linking results. We systematically compare several variants of SLSQL, each of which utilizes schema linking differently. We find that schema linking always leads to better SQL parsing performance. But the performance of schema linking is far from perfect, even under supervised learning ($\leq$ 0.83 F1). To maximize the potential of schema linking, we study how SLSQL performs during inference when provisioned with oracle schema linking results (i.e., ground truth annotation). As such, we find our simple SLSQL model performs impressively on Spider — the remaining gap is largely due to corpus noise, including inconsistent patterns and errors in the dataset. This evidence points to schema linking as a critical task for text-to-SQL parsing, and also provides an indicative upper bound on performance on the Spider dataset. Interestingly, our analyses on the failure cases caused by model deficiencies reveal advanced challenges on the text-to-SQL task like *deep logical reasoning* and *extremely complex structure* (*c.f.* Section 5). Our annotated data enables us to address these challenges without the interference of the current noisy schema linker.

In summary, we contribute an annotation of schema linking and in-depth analyses on the role of schema linking in the text-to-SQL task. We identify schema linking as a crux for the further improvement of text-to-SQL parsing. Our analyses provide insights to advance the understanding of text-to-SQL parsing, facilitating future research on the areas of problem identification, dataset construction and model evaluation.

2 Related Work

Text-to-SQL Parsing: Text-to-SQL parsing has been long studied in past decades (Finegan-Dollak et al., 2018; Yu et al., 2018c). Early text-to-SQL systems rely heavily on complicated rules and handcrafted feature engineering (Zhong et al., 2017; Finegan-Dollak et al., 2018). Fortunately, the research progress has been largely accelerated in recent years thanks to both large-scale text-to-SQL datasets (Zhong et al., 2017; Yu et al., 2018c) and interests in neural modeling (Xu et al., 2017; Dong and Lapata, 2018; Sun et al., 2018; Yu et al., 2018b; Guo et al., 2019; Wang et al., 2020). With years of studies, current research on this task focuses on addressing cross-domain generalizability and generating complex SQL queries. To improve cross-domain generalizability, advanced representations of the schema and the queries are explored, e.g., graph-based schema representations (Bogin et al., 2019b,a), contextualized question representations (Hwang et al., 2019; Guo et al., 2019) and relation-aware self-attention (Wang et al., 2020). As for the complex SQL query generation, approaches are proposed to constrain the output with SQL grammar, e.g., modular decoders for separate SQL clauses (Yu et al., 2018b), intermediate language representation (Guo et al., 2019), recursive decoding for nested queries (Lee, 2019), schema-dependent grammar for SQL decoding (Lin et al.,

2019), etc. Unlike their perspective, this work calls attention to schema linking, which we consider is the crux for the text-to-SQL task and yet to be sufficiently studied.

Schema Linking: The idea of schema linking has been broadly studied in similar tasks like entity linking in the field of knowledge graphs (Fu et al., 2020; Wu et al., 2019; Rijhwani et al., 2019; Logeswaran et al., 2019) and slot filling in dialogue systems (Xu and Hu, 2018; Ren et al., 2018; Nouri and Hosseini-Asl, 2018; Rastogi et al., 2017), where ample annotated data and models have been proposed to address their specific properties. In the general domain of semantic parsing, it has been demonstrated that decoupling underlying structure with lexicon benefits cross-domain semantic parsing (Su and Yan, 2017; Herzig and Berant, 2018). However, when it comes to the text-to-SQL problem, many existing approaches treat schema linking as a minor pre-processing procedure using simple heuristics, such as string matching between natural language utterances and column/table names (Guo et al., 2019; Yu et al., 2018a; Lin et al., 2019). As discussed in Dong et al. (2019), such simple heuristics are difficult to accurately identify columns/tables involved in a natural language utterance and well understand the relation between an utterance and the corresponding database schema. Therefore, they make the first step towards treating schema linking as an individual research problem. Nevertheless, due to the lack of direct schema linking supervision, they achieve limited improvement on the challenging Spider dataset, further illustrating the difficulties of this problem. Unlike these prior approaches, more recent models (Bogin et al., 2019a,b; Wang et al., 2020) integrate schema linking as a learnable component into the network, which brings significant improvements. In this work, we take one step further along this line to perform a thorough study by conducting schema linking annotation, discussing its importance and revealing its unique characteristics.

3 Schema Linking Annotation

To support a data-driven and systematical study, we annotate the schema linking information for each instance in the training and development set of Spider (Yu et al., 2018c) (the test set is hidden), the largest and most challenging text-to-SQL dataset so far. A simple way is conducting automatic annotation by matching table/column names and values in a ground truth SQL query against its corresponding natural language query. However, such automatic annotation method can bring much noise that would potentially hinder the model performance. Therefore, we annotate the dataset combining both automatic and manual processing.

Automatic Annotation: We first programmatically annotate the schema linking for easy cases to reduce the manual work. The string matching strategy we use is inspired by Guo et al. (2019). We first generate n-grams for each natural language query and only keep those with a length less than 6. For each condition value in a ground truth SQL query, we label the n-gram which it exactly matches. After labeling all n-grams matching with SQL condition values, we enumerate all unlabeled n-grams in the ascending order of length. If an n-gram contains all tokens in a column name, we label and regard this n-gram as a reference to that column. This process deals with cases where a column name is exactly mentioned or slightly paraphrased (e.g., column "type code" mentioned as "code of type"). However, a column/table name can also partially occur (e.g., column "type code" mentioned as "code"). To deal with such cases, we enumerate all unlabeled n-grams in descending order of length. If an n-gram contains any token in a column name, we label it as the reference to the corresponding column. Table reference labeling is conducted similarly.

Manual Annotation: We then recruit three computer science majors to further manually refine the automatic annotation. They are trained with a detailed annotation guideline and 50 trial samples. One is allowed to start after getting all trial samples correctly annotated. During the process, strict quality control is conducted by calculating their inter-annotator agreement (IAA)[2]. Specifically, the dataset is divided into 10 batches where instances in each batch are equally distributed to the annotators with 5% overlap. We accept a batch if the IAA is higher than 0.7 (at least 70% of the instances have exactly the same annotation by all three annotators); otherwise, the annotators are required to re-examine their annotations individually. Once a batch of annotation is accepted, we let the annotators discuss their disagreed annotations and come up with a final agreed result.

[2]In this work, we treat one instance as *agreed* if the three annotators have exactly the same annotations on it.

Category	Precision	Recall	F1
column	0.682	0.897	0.775
table	0.867	0.744	0.801
value	0.997	0.845	0.915

Table 1: The micro-average precision, recall and F1 of the automatic annotation using our final manual annotated data as the ground truth.

Analysis: To evaluate how noisy and coarse the automatic annotation is, we calculate its F1 score by treating the manually improved result as the ground truth (see Table 1). For value reference, some missing annotations are caused by abbreviation. For example, the text span "assistant professor" in a natural language query is abbreviated as `rank="AsstProf"` in the corresponding ground truth SQL query. For column and table reference, an issue is that it cannot deal with columns and tables are not textually referred to in the corresponding utterance. For example, the column `elevation` is mentioned as "altitude" in a sentence. Other issues of the automatic annotation are largely caused by similar column/table names.

4 Model

To conduct an in-depth study of the role of schema linking, we develop the SLSQL model, as depicted in Figure 2. It comprises a base model (the left part), which is based on the encoder–decoder structure, and an explicit schema linking component (the right part). While many sophisticated models have been proposed, we adopt this structure to leverage its simplicity to systematically analyze the factor of schema linking. In our experiments, we will study different variants of this structure by configuring the model in various ways.

Before the detailed model description, we first introduce our mathematical notations. We denote the natural language query Q as $Q = \{q_i\}_{i=1}^{|Q|}$, where q_i indicates the i^{th} token of Q and $|Q|$ indicates the length. Similarly, we denote the database schema E as $E = \{e_i\}_{i=1}^{|E|}$. It is the concatenation of a special token `[none]`, the name of each table and the names of its columns. Here, `[none]`, which we treat as a special element in the schema, is designed for schema linking — if a word in Q does not link to any column or table, it links to this token. In addition, we use MLP to mean multilayer perceptron, $\oplus$ to represent the concatenation operation, and bold symbols to denote dense representations.

4.1 Base Model

We now detail the base model which consists of an encoder and a decoder. The encoder (part ① in Figure 2) processes the input (i.e., Q and E) into hidden representation (denoted as $\boldsymbol{h}$) and the decoder (part ② in Figure 2) generates the SQL query (i.e, S) accordingly.

Encoder: Following (Hwang et al., 2019; Guo et al., 2019; Zhang et al., 2019), we concatenate the input query Q and database schema E to an integrated sequence as input for BERT (Devlin et al., 2019) to generate embeddings for each question token and element in the schema (namely $\boldsymbol{Q} = \{\boldsymbol{q_i}\}_{i=1}^{|Q|}$ and $\boldsymbol{E} = \{\boldsymbol{e_i}\}_{i=1}^{|E|}$) and the overall representation for the input as $\boldsymbol{h}$. Here, $\boldsymbol{E}$ consists of embeddings of all the columns/tables and the special token `[none]`. The embedding of the special token `[CLS]` in BERT is taken as $\boldsymbol{h}$. Formally, we have:

$$\{\texttt{[CLS]}, Q, E\} \rightarrow \boldsymbol{h}, \{\boldsymbol{q_i}\}_{i=1}^{|Q|}, \{\boldsymbol{e_i}\}_{i=1}^{|E|}. \quad (1)$$

Note that, in this representation, the schema linking information has also been captured by the multi-layer self-attention implicitly. However, we argue the explicit supervisions are required. While a plausible solution is to use the relation-aware encoding proposed by Wang et al. (2020) to do this, we later propose a simpler solution to facilitate our analytical study.

Decoder: Inspired by the prior work (Yin and Neubig, 2017; Dong and Lapata, 2016, 2018; Zhang et al., 2019), we adopt a two-step decoder to generate the SQL query from the hidden representation $\boldsymbol{h}$. We first generate a coarse SQL query S', namely a SQL sequence without aggregate functions, using a GRU network (Cho et al., 2014). We then synthesize the final SQL query S based on S'. The ② part in Figure 2 illustrates the generation of aggregate functions for the column `budget` during the decoding process.

4.2 Schema Linking Extension

To study the role of schema linking, we extend the encoder to explicitly capture the schema linking information. It works in two steps: in step 1.1, we learn the explicit schema linking based on our annotation; in step 1.2, we learn $\boldsymbol{h}$, the overall representation for the input, by integrating our explicit schema linking results.

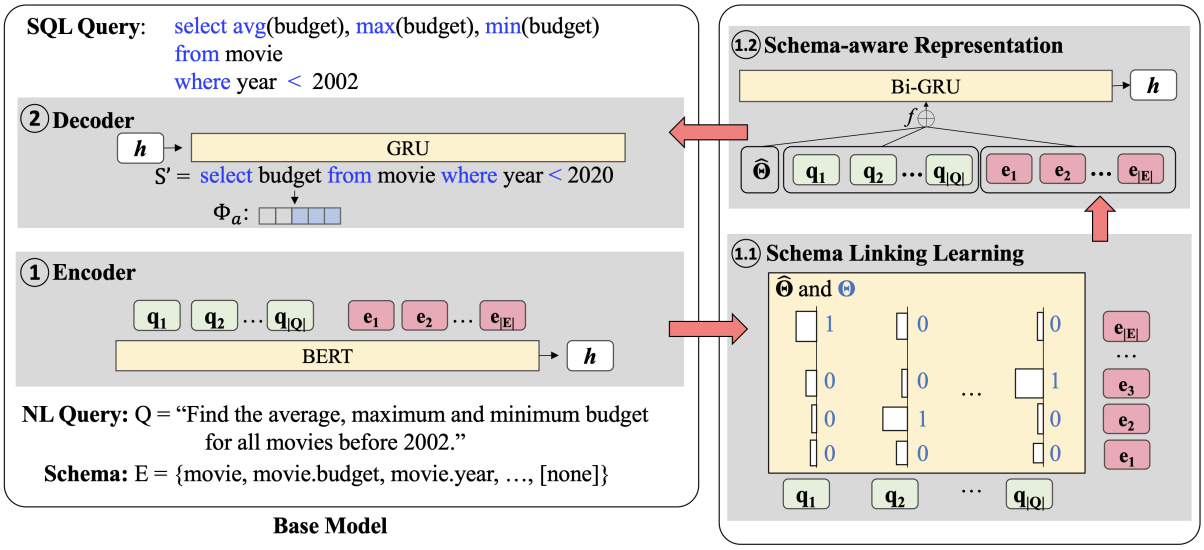

Figure 2: Schematic of our SLSQL model. The left part illustrates the base model, an encoder–decoder framework commonly used in the text-to-SQL task. The right part shows our extension of schema liking component.

Schema Linking Learning: We denote the ground truth schema linking distribution as Θ and the estimated one as $\hat{\Theta}$. We denote the linking probability of the token q_i and the schema element e_j as $\hat{\mathcal{P}}_{i,j}$, which is calculated as

$$\hat{\mathcal{P}}_{i,j} = softmax\big[MLP(\boldsymbol{q_i} \oplus \boldsymbol{e_j})\big] \tag{2}$$

where $1 \le i \le |Q|$ and $1 \le j \le |E|$. Therefore, the estimated schema linking distribution is formulated as $\hat{\Theta} = \{\hat{\mathcal{P}}_{i,j}\}_{1\le i\le|Q|,1\le j\le|E|}$. The loss function of the schema linking step is defined as

$$\mathcal{L} = -\frac{1}{|Q||E|}\sum_{i=1}^{|Q|}\sum_{j=1}^{|E|}\big[\mathcal{P}_{i,j}\log\hat{\mathcal{P}}_{i,j} + (1-\mathcal{P}_{i,j})\log(1-\hat{\mathcal{P}}_{i,j})\big], \tag{3}$$

where $\mathcal{P}_{i,j} \in \Theta$ is the ground truth value for each pair of q_i and e_j.

Schema-aware Representation: In step 1.2, we learn the schema-aware representation (i.e., $\boldsymbol{h}$) based on the predicted schema linking results (i.e., $\hat{\Theta}$), Q, and E. To cover the temporal relation, we use a bi-directional GRU to generate $\boldsymbol{h}$. Thus, the schema-aware representation is learned from

$$\boldsymbol{h} = \text{Bi-GRU}\big[f(\hat{\Theta}, \boldsymbol{Q}, \boldsymbol{E})\big], \tag{4}$$

where f is the reference mechanism. It is calculated by concatenating the embedding of q_i and the embedding of the schema element (including `[none]`) it likely links with. We here implemented it in a soft manner through weighted average embedding. Specifically,

$$f(\hat{\Theta}, \boldsymbol{Q}, \boldsymbol{E}) = \{\boldsymbol{q_i} \oplus \sum_{j=1}^{|E|}\hat{\mathcal{P}}_{i,j}\boldsymbol{e_j}\}_{i=1}^{|Q|}. \tag{5}$$

5 Experiments

In this section, we conduct systematic studies on the role of schema linking in text-to-SQL parsing. We examine several variants of SLSQL and thoroughly analyze the experimental results, providing detailed analyses and discussions to shed light on its unique characteristics.

5.1 Experiment Settings

Dataset: We conduct the experiments on the Spider (Yu et al., 2018c) dataset, a large-scale benchmark for cross-domain complex text-to-SQL task. Spider consists of 11,840 examples which are split into training (size: 7,000), development (size: 2,134) and test set (size: 1,034), covering 138 different domains. In addition, SQL queries in the dataset are categorized into four difficulty levels based on the number of SQL keywords. Models are evaluated using the official *exact matching accuracy* metric of Spider. We conduct ablation studies on the development set since the test set is used for scoring models on the leaderboard and is not publicly accessible.

Model Variants: To study the contribution of schema linking to the text-to-SQL parsing problem, we examine the following variants of SLSQL,

each with one way of utilizing the schema linking information. To facilitate discussions, the model described in Section 4 is referred to as **default** in this section.

- **base model:** To evaluate the impact of explicit schema linking, we use the *base* model, as a variant, whose encoder is defined as EQ. (1) and followed by the decoder directly. Note there is no schema linking component in this variant.
- **auto:** To validate the advance of the manual annotation, we build this variant, which is trained with the automatic schema linking annotation in EQ. (3) instead of the manual annotation.
- **hard reference**: As introduced in Section 4.2, we generate schema-aware representation using reference mechanism f. Here, we further build this variant using hard reference concatenation. Specifically, instead of computing weighted average embeddings of all database schema elements as described in EQ. (5), we concatenate a question token embedding $\boldsymbol{q_i}$ with $\boldsymbol{e_j}$ which is the embedding of the schema element with the highest linking probability, i.e. $\hat{\mathcal{P}}_{i,j}$, to validate whether a different way of integrating schema linking information impacts the performance.
- **oracle:** To explore the maximum potential benefit of schema linking to the text-to-SQL task, we design the *oracle* variant. In this variant, we remove the schema liking learning component (step 1.1 in Figure 2), connect the encoder part with step 1.2 directly, and replace the estimated distribution $\hat{\Theta}$ with the ground truth distribution Θ in EQ. (4), namely, $\boldsymbol{h} = \text{Bi-GRU}\big[f(\Theta, \boldsymbol{Q}, \boldsymbol{E})\big]$. Like the *auto* variant, we build the *oracle_auto* variant using the automatic annotation instead.

Implementation Details: We use Stanford CoreNLP (Manning et al., 2014) to preprocess the corpus. We implement SLSQL in PyTorch (Paszke et al., 2017). We use the pre-trained uncased BERT-Base model with 12 layers provided by Wolf et al. (2019). We use Adam (Kingma and Ba, 2014) with the learning rate set to 5×10^{-5} and batch size set to 4. Considering the ablation studies have to be conducted on the development set due to the model submission policy of Spider (at most 2 models are allowed for evaluation on the hidden test set), all hyperparameters are tuned on the training set. The model converges within 20 epochs.

Model		SL	Dev	Test
SLSQL	default	0.81	61.4	55.0
	hard_ref	0.80	60.8	55.7
	base	-	57.4	-
	auto	0.77	59.2	-
	oracle	1.0	**72.4**	-
	oracle_auto	0.83	65.7	-
GlobalGNN (Bogin et al., 2019b)		-	52.7	47.4
EditSQL (Zhang et al., 2019)		-	57.6	53.4
IRNet (Guo et al., 2019)		-	61.9	54.7
Bertrand-DR (Kelkar et al., 2020)		-	57.9	54.6
RYANSQL (Choi et al., 2020)		-	70.6	60.6
RATSQL (Wang et al., 2020)		-	69.7	**65.6**

Table 2: Exact matching accuracy of our model variants and other recently published text-to-SQL models which are evaluated on Spider. The *default* and *hard_ref* variants are evaluated on the hidden test set. For each variant, we report their overall schema linking F1 score (denoted as **SL**) on the development set by treating the manual annotation as the ground truth.

5.2 Overall Performance Analysis

Table 2 shows the performance of SLSQL and other recent models evaluated on Spider. We can observe that there is a strong positive correlation between the schema linking performance and the exact matching accuracy. For example, without schema linking, SLSQL-*base* has the lowest exact matching accuracy among all variants. By comparing the three variants with a trained schema linking component (*default*, *hard_ref* and *auto*), we find that training SLSQL models with higher quality annotations leads to better performance (*default* & *hard_ref* v.s. *auto*), while using soft or hard concatenation for propagating schema linking information does not make a significant difference (*default* v.s. *hard_ref*). As expected, feeding SLSQL with Θ rather than $\hat{\Theta}$ during inference leads to a significantly higher result (*oracle* and *oracle_auto*). Similarly, using the manual annotation instead of the automatic annotation for the oracle setting largely improves the model performance.

We also list top models on the Spider leaderboard for reference in Table 2. As we do not aim at putting effort in building sophisticated models, our *default* and *hard_ref* variants cannot compete with the state-of-the-art models like RYANSQL (Choi et al., 2020) and RATSQL (Wang et al., 2020). However, the *oracle* variant shows that a simple model has the potential to sharply outperform these strong models (on the development set), by improving schema linking performance, which shows an important future direction. We will have further discussions on this issue in Section 5.5.

Variant	easy	medium	hard	extra	all
default	79.6	66.6	49.4	33.5	61.4
hard_ref	79.2	66.4	48.9	31.8	60.8
base	77.6	60.0	47.1	31.8	57.4
auto	77.2	64.1	48.3	31.2	59.2
oracle	**89.5**	**75.1**	**66.1**	**46.4**	**72.4**
oracle_auto	86.0	70.2	53.4	36.5	65.7

Table 3: Exact matching accuracy by difficulty on the development set of Spider.

Reference	Precision	Recall	F1
column	0.826	0.820	0.823
table	0.806	0.840	0.822
value	0.773	0.741	0.757

Table 4: Schema linking results of different references categories on the development set of Spider. Precision, recall and F1 scores are micro-averaged.

World Knowledge	Ex.1	Q: How many languages are spoken in Aruba? G: `country.name` - *value* P: `countrylanguage.language` - *value*
	Ex.2	Q: ...in African countries that are republics? G: `country.government_form` - *colum* P: *none*
Semantic Understanding	Ex.3	Q: ...average rank for winners in all matches? G: `matches.winner_rank` - *column* P: `ranking.rank` - *column*
	Ex.4	Q: ...names of students who have no friends? G: `highschooler` - *table* P: *none*
Type Error	Ex.5	Q: List all the student details ... G: `student.other_student_details` - *column* P: `student` - *table*

Table 5: Representative erroneous schema linking predictions. Notations Q, G, P stand for question, ground truth and prediction, respectively. Reference types are in italics and *none* means not being a reference.

5.3 Can Schema Linking Help Manage Complex Queries?

To have further insights on how schema linking can help complex queries, we investigate detailed model performance under different difficulty levels. Table 3 presents the result. We can observe that, generally schema linking helps boost the performance of SLSQL across all different difficulty levels (*base* v.s. other variants). More accurate schema linking predictions lead to more significant accuracy improvements. For example, the *oracle* variant, which has access to the manual annotation, achieves the highest score on all different difficulty levels. Besides, *default* and *hard_ref* outperform the *auto* variant trained with automatic annotation on all difficulty levels, thanks to the higher quality of schema linking annotation.

5.4 Schema Linking Performance Analyses

To have a better understanding of the schema linking task itself, we test its performance for SLSQL-*default* and list some representative wrong predictions, as shown in Table 4 and Table 5. We observe that, with the model trained with explicit supervision, the F1 scores for column, table and value linking are still far from satisfactory, demonstrating that schema linking is not an easy task and requires future efforts to improve.

Particularly, linking value references with the schema is the most difficult part as its F1 score is the lowest. We find most of wrong value predictions are due to the lack of world knowledge. As shown in Ex.1, the model mistakenly predicts Aruba as a language instead of a country. In Ex.2, the model fails to understand that "republic" is a government form. We find that, despite using BERT as underlying language understanding module, the model still has difficulty in dealing with some of such value reference linking. To help the model accurately link value references with the schema, a solution can be scanning the content stored in the database, as applied in some prior work (Bogin et al., 2019b; Wang et al., 2020), to facilitate the model inference. The motivation behind it is that "Aruba" occurs in the column `country.name` instead of `countrylangauge.language` or `city.name`. However, in real scenarios, database contents are not always accessible to text-to-SQL models and condition values mentioned in human utterances do not necessarily exist in the database (Zhong et al., 2017). An alternative solution can be looking for external knowledge resources which easily identify that the word "republic" is related to "government form", as adopted in Guo et al. (2019). Nevertheless, such solution relies heavily on the quality and availability of knowledge resources, making the model less portable for practical use. Most of the remaining errors have commonalities with wrong table/column reference linking, in terms of causes.

When it comes to column and table reference linking, we find the error sources are complex, which mainly include failing to capture semantic relations between words and tables/columns and predicting other linking types. We observe that sometimes the model is biased towards predicting columns/tables that exactly occur in an utterance while neglecting more global information. As

shown in Ex.3, the model links the word "rank" to the column `ranking.rank` while the correct choice should be `matches.winner_rank` if considering more global information. In addition to neglecting global semantic information, failing to capture semantic similarity between words is another cause of errors such as the case shown in Ex.4. We find that some of such issues are caused by the WordPiece tokenization (Wu et al., 2016) in BERT. For example, it tokenizes "highschooler" as "highs ##cho ##ole ##r" while tokenizing "high schooler" as "high school ##er", which is the cause of this case. Besides, columns can have common words with their table names, making the model mistakenly predict some part of a column reference as a table reference (or vice versa) in some cases like Ex.5. However, with such linking errors, the model usually is still able to finally generate desired SQL queries, which means they typically are not as harmful as other errors described above.

Fortunately, many of the aforementioned problems have been extensively studied in similar tasks such as zero-shot entity linking in knowledge graph tasks (Wu et al., 2019; Logeswaran et al., 2019; Rijhwani et al., 2019; Fu et al., 2020) and domain adaptive slot filling in dialogue system domains (Xu and Hu, 2018; Rastogi et al., 2017; Ren et al., 2018; Nouri and Hosseini-Asl, 2018). With annotated data, there is an ample room to transfer these approaches, which are mostly based on supervised learning, to the schema linking problem for further improving the text-to-SQL parsing ability.

5.5 Error Analysis of the Oracle Variant

To further investigate what the remaining problems lie on, provided that schema linking can be done perfectly, we conduct error analyses using the *oracle* variant. We randomly sample 100 error instances on the development set. We analyze the errors and classify them into three categories: *Correct Equivalent*, *Corpus Error* and *Model Incapability*. Considering many examples in the dataset have textually similar questions, erroneous predictions having the same cause are counted once during analysis. This process is repeated for five times and we take the average percentage for each error type. We find many errors are due to the the corpus noises, namely the first two error types. Table 6 provides representative examples for each category. We now detail the three error categories.

Correct Equivalent: One SQL query can have several semantic equivalents with different writing patterns. We find that in the Spider dataset, it is not always consistent that which of such patterns is given as the ground truth. We identify some SQL queries generated by SLSQL are actually semantically correct while treated as wrong predictions due to not only co-existence of different SQL writing patterns in the training set but also the exact matching evaluation. According to our manual verification, such false negative samples take up around 30% of the sampled errors. Ex.1 shows a case where two SQL queries are semantically equivalent, despite different writing patterns. There are also some inconsistent patterns in particular clauses like `group by`, as illustrated in Ex.2. Such errors suggest that either pattern consistency or more flexible, robust evaluation metric should be focused on for future dataset construction.

Corpus Error: After carefully examining each sampled errors, we also identify around 26% of them are caused by incorrectly annotated example in the dataset, e.g., wrong ground truth SQL queries, incomprehensible utterances, problematic database schemas, etc. As shown in Ex.3, "greater area than that of any country" is indeed logically equivalent to "greater than the maximum area of all countries", while the ground truth SQL query means "greater area than that of some countries". Some of these errors are even hard to identify as incorrect ground truth at first glance. Ex.4 looks like a correct equivalent case while actually the ground truth SQL query is wrong. Moreover, we find typos in natural language queries can lead to incorrect SQL queries. For example, one instance has a text span "the sname of every sing" in the natural language query which we believe should be "the name of every song". While better data annotations definitely result in better SQL parsing performance, such errors suggest a robust text-to-SQL parser should be tolerant of noises like typos and grammatical errors, which is an important but overlooked problem for real application.

Model Incapability: Even with oracle schema linking annotation, SLSQL is yet to be perfect. We find about 44% of the failing instances are due to modeling incapability. Many problems lie in the requirement of *deep logical reasoning* and *extremely complex structure*. Considering the case shown in Ex.5, the model directly translates the word "and"

Correct Equivalent (29.6%)	Ex.1	Q: What are the names of people who **do not play poker**? G: select name from people where people_id **not in** (select people_id from poker_player) P: select people.name from people **except** select people.name from poker_player join people
	Ex.2	Q: For **each shop**, return the number of employees working there and the name of the shop. G: select count(∗), t2.name from hiring as t1 join shop as t2 on t1.shop_id = t2.shop_id group by t2.**name** P: select shop.name, count(∗) from shop join hiring on hiring.shop_id = shop.shop_id group by shop.**shop_id**
Corpus Error (26.3%)	Ex.3	Q: Which countries have **greater area than that of any country** in Europe? G: select name from country where surface_area > (select **min**(surface_area) from country where continent = 'Europe') P: select name from country where surface_area > (select **max**(surface_area) from country where continent = 'Europe')
	Ex.4	Q Find the number of concerts happened in the stadium **with the highest capacity**. G: select **count**(∗) from concert as t1 join stadium as t2 order by t2.capacity **desc limit 1** P: select count(∗) from stadium join concert where stadium.capacity = (select max(capacity) from stadium)
Model Incapability (44.1%)	Ex.5	Q: What is the total surface area of the continents Asia **and** Europe? G: select sum(surface_area) from country where continent = 'Asia' **or** continent = 'Europe' P: select sum(surface_area) from country where continent = 'Asia' **and** continent = 'Europe'
	Ex.6	Q: How many countries speak **both** English and Dutch? G: select count(∗) from (select t1.name from country as t1 join countrylanguage as t2 where t2.language = 'English' intersect select t1.name from country as t1 join countrylanguage as t2 where t2.language = 'Dutch') P: select count(∗) from countrylanguage where countrylanguage.language = 'English' intersect select count(∗) from countrylanguage where countrylanguage.language = 'Dutch'

Table 6: Representative examples of the three error types and their average percentages during sampling. Notations Q, G and P stand for question, ground truth and prediction, respectively. Some `on` clauses are omitted for display.

into the SQL keyword *and*, leading to a SQL query with contradictory conditions. Although this query is classified as "medium" by the Spider evaluation script, it is actually difficult as it requires a model to perform logic reasoning based on the understanding that a country cannot be in Asia and Europe at the same time. A similar case is "singers with birth year before 1945 and after 1955" where a numeric comparison is required to avoid generating contradictory `where` conditions. Ex.6 is a case of extremely complex structure where three logical steps are required to synthesize the SQL query, i.e., 1) selecting English-speaking countries and Dutch-speaking countries; 2) finding their intersection using `intersect`; and 3) counting the intersection size with an outer query. Unfortunately, the model writes a plausible but wrong SQL query.

Discussion: The above results and analyses suggest that, with schema linking well solved, even a simple BERT baseline can capture quite a large portion of the patterns in the Spider dataset. This indicates that schema linking is the crux for current research on text-to-SQL task, providing an appealing perspective to this task. Also, through experiments and analyses with schema linking annotation, some previously unnoticed challenges like *deep logical reasoning* and *extremely complex structure* have emerged, also pointing further research directions. Such problems were interwoven with schema linking problems in the original Spider dataset. Our schema linking annotation makes it possible for such problems to be separately approached without the interference of database schemas.

6 Conclusion

We critically examine the role of schema linking for the text-to-SQL task. To support model-independent and thorough studies, we invest human resources to annotate schema references and contribute a high-quality, large-scale schema linking corpus. Experimenting with our designed Schema Linking SQL (SLSQL) model, we demonstrate that more accurate schema linking conclusively leads to better text-to-SQL parsing performance. Importantly, given oracular schema references, a simple BERT model like SLSQL can achieve an impressive performance. Our experiments show that schema linking, often overlooked as simple preprocessing, is actually a requisite for good SQL parsing performance, providing an intriguing perspective for future improvements on this task. Our study sheds light on the characteristics of text-to-SQL parsing for future efforts including advanced modeling, problem identification, dataset construction and model evaluation.

Acknowledgments

This research is supported by the National Research Foundation, Singapore under its International Research Centres in Singapore Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. We thank the anonymous reviewers for their precious comments. We also thank Bo Pang and Tao Yu for helping with the model evaluation.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019a. Representing schema structure with graph neural networks for text-to-SQL parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565, Florence, Italy. Association for Computational Linguistics.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019b. Global reasoning over database structures for text-to-SQL parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3657–3662, Hong Kong, China. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. *arXiv preprint arXiv:2004.03125*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Zhen Dong, Shizhao Sun, Hongzhi Liu, Jian-Guang Lou, and Dongmei Zhang. 2019. Data-anonymous encoding for text-to-SQL generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5404–5413, Hong Kong, China. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Xingyu Fu, Weijia Shi, Zian Zhao, Xiaodong Yu, and Dan Roth. 2020. Design challenges for low-resource cross-lingual entity linking. *arXiv preprint arXiv:2005.00692*.

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 *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2018. Decoupling structure and lexicon for zero-shot semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1619–1629, Brussels, Belgium. Association for Computational Linguistics.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Amol Kelkar, Rohan Relan, Vaishali Bhardwaj, Saurabh Vaichal, and Peter Relan. 2020. Bertrand-dr: Improving text-to-sql using a discriminative re-ranker. *arXiv preprint arXiv:2002.00557*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Dongjun Lee. 2019. Clause-wise and recursive decoding for complex and cross-domain text-to-SQL generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6044–6050, Hong Kong, China. Association for Computational Linguistics.

Kevin Lin, Ben Bogin, Mark Neumann, Jonathan Berant, and Matt Gardner. 2019. Grammar-based neural text-to-sql generation. *arXiv preprint arXiv:1905.13326*.

Lajanugen Logeswaran, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, Jacob Devlin, and Honglak Lee. 2019. Zero-shot entity linking by reading entity descriptions. *arXiv preprint arXiv:1906.07348*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

Elnaz Nouri and Ehsan Hosseini-Asl. 2018. Toward scalable neural dialogue state tracking model. *arXiv preprint arXiv:1812.00899*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

A. Rastogi, D. Hakkani-Tr, and L. Heck. 2017. Scalable multi-domain dialogue state tracking. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 561–568.

Liliang Ren, Kaige Xie, Lu Chen, and Kai Yu. 2018. Towards universal dialogue state tracking. *arXiv preprint arXiv:1810.09587*.

Shruti Rijhwani, Jiateng Xie, Graham Neubig, and Jaime Carbonell. 2019. Zero-shot neural transfer for cross-lingual entity linking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6924–6931.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1235–1246, Copenhagen, Denmark. Association for Computational Linguistics.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware SQL generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 361–372, Melbourne, Australia. Association for Computational Linguistics.

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 *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2019. Zero-shot entity linking with dense entity retrieval. *arXiv preprint arXiv:1911.03814*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Puyang Xu and Qi Hu. 2018. An end-to-end approach for handling unknown slot values in dialogue state tracking. *arXiv preprint arXiv:1805.01555*.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594, New Orleans, Louisiana. Association for Computational Linguistics.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018b. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

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. 2018c. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

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 *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5338–5349, Hong Kong, China. Association for Computational Linguistics.

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

A Model Details

A.1 Encoder Implementation

For each example, we concatenate all the column/table names, a special token `[none]` and the natural language query, separated by `[SEP]`, as the input sequence to BERT. Here the special token `[none]` is designed for the subsequent schema reference resolution that we will introduce later. For each column, we concatenate the column name with its table name, separated by a dedicated symbol `[#]`, as its canonical string representation. If a word is tokenized into multiple pieces by the BERT tokenizer, the embedding of the first piece is taken as the corresponding word embedding. Representations of columns and tables are computed through a GRU (Cho et al., 2014) which takes as input their word embeddings generated by BERT.

A.2 Decoder Implementation

The decoder of SLSQL is largely based on the work of Zhang et al. (2019) and uses the attention mechanism (Bahdanau et al., 2014; Luong et al., 2015). For ease of readability, we define the attention function $attention(\mathrm{Q}, \mathrm{K})$ as follows:

$$\begin{aligned} \text{score}_i &= \mathbf{Q}\mathbf{W}\mathbf{K}_i \\ \alpha &= softmax(\text{score}) \\ \text{output} &= \sum_i \alpha_i \mathbf{K}_i \end{aligned} \tag{6}$$

where $\mathbf{W}$ is a learnable weight. At each decoding step t, we generate the next hidden state $\mathbf{h}_{t+1}$ as follows:

$$\mathbf{h}_{t+1} = \text{GRU}\big([\mathbf{k}_t; \mathbf{s}_t; \mathbf{c}_t]\big) \tag{7}$$

where $\mathbf{k}_t$ is the embedding of the clause keyword at step t, which can be `select`, `from`, `where`, etc. We denote the embedding of the generated SQL query token at step t as $\mathbf{s}_t$. The context vector $\mathbf{c}_t$ is the concatenation the context of column/table and the context of natural language question.

$$\mathbf{c}_t = [\mathbf{c}_t^{\text{col/tbl}}; \mathbf{c}_t^{\text{q}}] \tag{8}$$

Here, the context of column/table $\mathbf{c}_t^{\text{col/tbl}}$ is obtained as follows:

$$\begin{aligned} \mathbf{e}_{i,j}^{\text{q}} &= [\mathbf{e}_i; \mathbf{q}_j; \mathbf{f}_i] \\ \tilde{\mathbf{e}}_i^{\text{q}} &= attention(\mathbf{k}_{t-1}, \mathbf{e}_i^{\text{q}}) \\ \mathbf{c}_t^{\text{col/tbl}} &= attention(\mathbf{h}_t, \tilde{\mathbf{e}}^{\text{q}}) \end{aligned} \tag{9}$$

where $\mathbf{e}_i$ is the embedding of i-th column/table. Here, $\mathbf{f}_i$ is a feature vector that consists of binary features such indicating whether i-th column/table is a primary key or a foreign key column, etc. We have the context of question as follows:

$$\mathbf{c}_t^{\text{q}} = attention(\mathbf{h}_t, \tilde{\mathbf{q}}) \tag{10}$$

where $\tilde{\mathbf{q}}$ is the reference-aware question representation. At the decoding step t, the SQL query token $\mathbf{y}_t$ is predicted as follows.

$$\begin{aligned} \text{score}_i^{\text{col/tbl}} &= MLP\big([\mathbf{h}_t; \tilde{\mathbf{e}}_i^{\text{q}}; \mathbf{c}_t^{\text{q}}]\big) \\ \text{score}^{\text{kw}} &= MLP\big([\mathbf{h}_t; \mathbf{c}_t^{\text{q}}]\big) \\ P(\mathbf{y}_t) &= softmax\big([\text{score}^{\text{col/tbl}}; \text{score}^{\text{kw}}]\big) \end{aligned} \tag{11}$$

If $\mathbf{y}_t$ is a clause keyword (e.g., `select`, `from`, `where`, etc.), we set $\mathbf{k}_{t+1}$ as the embedding of $\mathbf{y}_t$. Otherwise, it will remain as is. If $\mathbf{y}_t$ is a column in `select`, `group by`, or `having` clause, we predict its aggregate functions as follows:

$$\begin{aligned} \text{score}_{\text{agg}} &= MLP\big([\mathbf{h}_t; \tilde{\mathbf{e}}_i^{\text{q}}]\big) \in \mathbb{R}^5 \\ P_{\text{agg}}(j) &= sigmoid\big(\text{score}_j^{\text{agg}}\big) \end{aligned} \tag{12}$$

where j is the index of the five aggregate functions.

A.3 Inference Constraints

During inference, syntax-based constraints are applied to prune the prediction space. For example, `having` can never come before `group by`. Since columns with aggregate functions are reduced into a single token in the target SQL sequence, we can determine the type of the next token before it is generated. For example, a column comes after a `select` token or a comma (`,`). To this end, a transition function constructed by scanning over the dataset is used to further prune the prediction space in Formula 11.