Improving text-to-SQL conversion by a effective and efficient schema linking method and SQL error correction

Chuxiang Luo

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

Language models have achieved remarkable performance on the task of translating natural language questions to SQL queries which can be used to extract data from database. Previous works that employs small-size language models emphasise on improving the SQL generation capability of the model, ignoring the importance of schema linking and error correction. This project proposes a effective and efficient schema linking method based on ColBERT retrieval model to find the most relevant tables and columns for the question and a simple SQL error correction method to fix execution errors in generated SQL queries. The BIRD benchmark is used to conduct a comprehensive evaluation which shows both methods can improve the performance of text-to-SQL.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Chuxiang Luo*)

Acknowledgements

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Translating a natural language question from users to a executable Structured Query Language (SQL) query is called text-to-SQL task [44]. It can be used across various fields, such as healthcare [56] and finance [61], to assist users that lack database knowledge and SQL programming skills in interacting with database and extracting expected data by human language. Lots of researches on text-to-SQL task utilize pretrained language models of small size. Li et al. [23] and Shin et al. [49] developed text-to-SQL models by fine tuning T5 model [45] that has 3 billion parameters and BART [22] that has 406 million parameters. Scholak, Schucher and Bahdanau [47] fine tuned T5 model with a incremental parsing that constraints SQL generation. These methods can be deployed locally, which avoid uploading data to external servers that may cause data leakage issues. In addition, the inference of small-size model takes a short time. However, their effectiveness are lower than those methods [13, 30] that use large language models like GPT-4 [1] which has more than one trillion parameters.

1.2 Problem

Text-to-SQL task can be divided into two parts: schema linking and SQL query generation. The meaning of schema linking is mapping words in the question to tables and columns in the database schema [52]. As shown in Figure 1.1, Given a question ”Name movie titles released in year 1945. Sort the listing by the descending order of movie popularity.”. Movie should be mapped to the table ”movies”. Movie titles and movie popularity should refer to the column ”movies.title” and ”movies.popularity”, respectively. A model can then use identified tables and columns to generate SQL query. Schema linking can reduce the noise in the input of the model and it is critical for reducing the performance gap on text-to-SQL between small-size language models and large language model, even if the model has a simple architecture [21]. While there are a large number of researches focus on accurately identifying the relevance between questions and schema items [21, 23, 36, 43, 54], these methods have not reached perfect schema linking especially for complex database and the efficiency of schema linking has been ignored. In addition, compared to the studies on improving SQL generation, fixing generated SQL queries have been less explored.

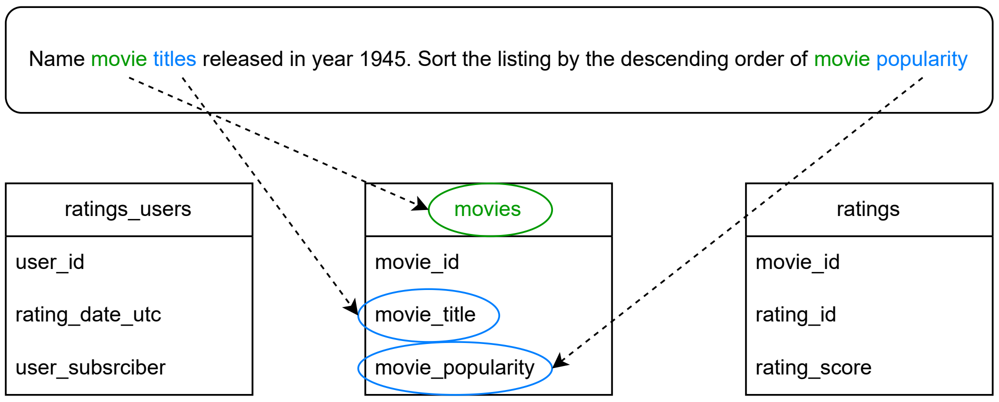


Figure 1.1: An example of schema linking, which has one question and three tables. Each table has three columns.

1.3 Objectives and Contributions

This project aims to improve the text-to-SQL performance of small-size language models by using a effective and efficient retrieval approach to find the most relevant tables and columns from the database schema to the question and applying SQL error correction to avoid execution errors. Specifically, this project will explore the application of ColBERT retrieval architecture [20] which not only provides interaction across its inputs but also allows embedding pre-computing. Furthermore, the SQL error correction uses simple approach that does not involve language models, which prevents an increase in required GPU memory and inference time.

The project's contributions to the text-to-SQL field are shown as follows:

- Implemented and trained schema linking model based on ColBERT, targeting accuracy and fast relevance identification.

- Through evaluating the schema linking model solely and jointly with SQL generation models with different sizes, the results shows it can achieve higher performance compared to existing methods.
- Investigated the influence of various ranking methods on the performance of schema linking.
- Implemented and evaluated error correction for generated SQL queries and the evaluation results proves it can slightly improve the performance of text-to-SQL.

The following section is structured as follows.

- Chapter 2: provides a detailed description of background knowledge that is related to text-to-SQL task, including database schema, different methods for text-to-SQL, useful external knowledge, schema linking, SQL error correction, retrieval models and approaches.
- Chapter 3: explains the model architecture of schema linking and the whole workflow of text-to-SQL.
- Chapter 4: dives into the implementation of each part of text-to-SQL and mentions the training details for the schema linking model.
- Chapter 5: includes the evaluation for schema linking model and the entire text-to-SQL and analyses the reasons for the changes in performance.
- Chapter 6: summarizes the ideas and important findings for this project as well as describes the limitation and the suggestions of future work.

Chapter 2

Background

2.1 Database Schema

In Figure 1.1, there is a simple database schema that only contains few tables and columns. In the real application, tables and columns are numerous. In addition, other meta information are contained in the database schema, which are critical for understanding database structure and data meaning.

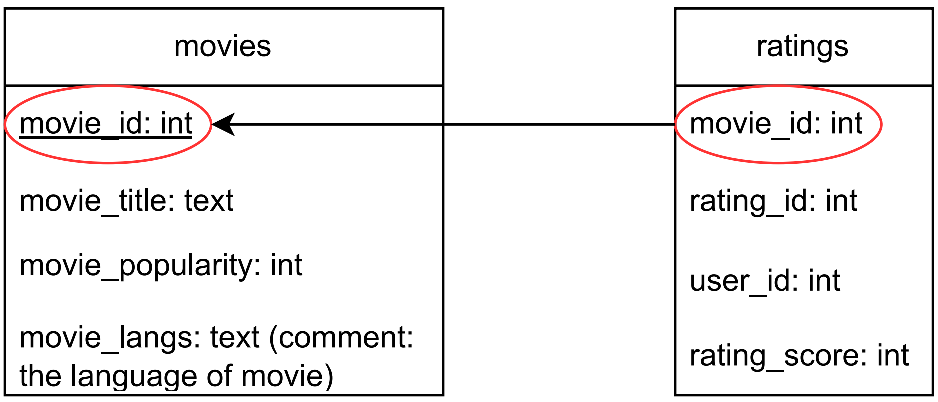


Figure 2.1: An illustration of database schema with meta information. Underline means the column is a primary key. Arrow line points from a foreign key to the column it refer to.

Common meta information includes:

- Primary key: a column or a set of columns has unique value for each row in the table, which serves as a unique identifier. For instance, ”movie id” is the primary key for the table ”movie”.

- Foreign key: a column refer to another table's column whose values are unique, which builds the relationship between two tables. Although it is not mandatory, a foreign keys is often a primary key in another table. In Figure 2.1, both tables has ”movie id”, ”movie id” in the table ”ratings” is the foreign key that point to ”movie id” in the table ”movie”.
- Data constraints: database avoids invalid values by adding constraints, such as data type (e.g. integer and text), values should be unique and values should not be null.
- Comments: tables and columns can have comments which provide a clear description for them. For instance, in the table ”movie” from Figure 2.1, the column ”movie langs” uses abbreviation in the name. A comment is required to understand the meaning of column and the information it stores.
- Representative values: some values in the column that are representative. For example, the values ”regular” and ”large” in the column ”drink size”.

2.2 Text-to-SQL models

According to the size of models, there are two kind of methods for developing text-to-SQL models: fine tuning and prompting. Fine tuning changes the value of model parameters, which result in more accurate predictions from the model. It is usually used for small-size models, including traditional deep neural networks (e.g. LSTM and GNN) and transformer-based models with less than 15 billion parameters. For large language models like LLaMA-70B [53], fine tuning takes a large amount of time and computational resources, which is not acceptable. Instead, prompting [35] which only changes the input to guide the generation of models is more suitable for large language models.

2.2.1 Fine-Tuning-Based Methods

Prior works focus on fine tuning the models with encode-decoder architecture. IRNet [15] employs two separate bi-directional LSTMs (BiLSTM) to encode question and schema and then concatenates them as the input of LSTM decoder. The encoder architecture of EditSQL [62] is more complex. A column header, which consist of table name and column name, is input to the first BiLSTM to obtain the initial representation.

After using attention mechanism to model the relationship between columns and the relationship between the question and columns, the second BiLSTM over all column headers generates the final column representations. Furthermore, some researches utilize BERT [9] to the encoder due to its powerful representation learning capability. SLSQL [21] uses the concatenation of question and schema as the input of BERT. The encoder of BRIDGE [33] also contains BiLSTM that subsequently processes the output of BERT for enhancing question and schema representations. In addtion, the features of meta information (e.g. primary keys) are incorporated into schema representations by a feedfoward network.

The studies mentioned above develop decoder based on LSTM or GRU [8] which is a variant of LSTM. More recently, transformer-based models like T5 [45] are used for both encoding and decoding. Rai et al. [46] leverages T5 model with component boundary marking which adds special tokens into question and SQL query to mark aligned segments, and token processing which involves replacing abbreviation of SQL keyword with full name and adding space for schema items and column reference in SQL query. PICARD [47] introduces a constrained decoding strategy that applies some checks for tokens and a score of negative infinite will be given to tokens that fail in the checks, whereas RESDSQL [23] treats SQL query generation as a two-stages process. The decoder firstly produces the skeleton of the SQL query that only has SQL keywords (e.g. SELECT, FROM and WHERE), and then generate the completed SQL query by filling the skeleton. T5-SR [28] reduces the mismatch between natural language question and SQL query through the intermediate representation and uses a re-ranker model to pick the best result from the candidates of generated SQL queries.

Instead of encoding text information for question and schema, graphs can be used to capture the question structure and relations in the database. Schema-GNN [4] construct the schema graph by representing columns and tables as nodes and relations (e.g. foreign-primary keys and column-belong-to-table) as edges. Then, graph neural networks are used to learn representation for schema. Global-GNN [3] is similar to Schema-GNN [4] but uses graph convolutional network for schema encoding and computes relevance between question and graph nodes. SADGA [6] represents question and schema by two separate graphs. The edges in the question graph include word distance dependency and parsing-based dependency which is useful to learn grammar. Graphix-T5 [25] displays the concatenation of question and schema as a single graph and learns structural information from the graph via GNNs.

In addition to encoder-decoder architecture, decoder-only architecture has been attended due to the success of large language models. DTS-SQL [43] fine tunes DeepSeek-7B [2] which is a general language models for various tasks. In contrast, Codes [24] is built on StarCoder [27] which is a language model specified for code generation. In particular, it adopts bi-directional data augmentation (Question-to-SQL and SQL-to-Question) to synthesize authentic data and thus improving the model generalization.

2.2.2 Prompting-Based Methods

Prompting is a technique that modifies the input to improve the performance of models on specific tasks. ChatGPT-SQL [34] contains the question, the entire schema and instructions in the prompt, while C3 [10] which is also based on ChatGPT only keeps most relevant schema items. Prompting used in these two models is zero-shot since they have not provide some examples to the model. Adding some (Question, true SQL query) demonstrations to create few-shot prompt is helpful, which is used in SQL-PaLM [50]. PET-SQL [30] introduces a two-stage framework which produces a preliminary SQL by few-shot prompting, uses entities from the first SQL for schema linking and then generates the final SQL and then the final SQL is produced with the linked schema. Furthermore, since the reasoning the reasoning ability of model can be enhanced by chain-of-thought (CoT) prompting which has intermediate reasoning steps [57], Divide-and-Prompt [37] decomposes text-to-SQL into some sub-tasks and uses GPT-3.5 with CoT prompting to address each sub-task. MAC-SQL [55] not only divides text-to-SQL into sub-tasks but also considers the question as simpler sub-questions and generates SQL query for each sub-question through CoT prompting.

2.3 External Knowledge Evidence

Some questions are hard for text-to-SQL without external knowledge. For instance, given the question ”List all movies with the best rating score”, it is necessary to know the definition of the best rating score (e.g. best rating score means rating_score = 5) for correct SQL query generation. ReGrouP [11] retrieves relevant knowledge from a created general knowledge bank, while each question in BIRD [26] has known external knowledge. Both methods have shown the effectiveness of external knowledge on text-to-SQL. In contrast to human-collected knowledge, Knowledge-to-SQL [17] utilizes language models to automatically produce knowledge given the question and the database schema. Compared with SQL query generation without knowledge, using machine-generated knowledge can achieve higher performance. However, its effectiveness remains inferior to the effectiveness of using human-collected knowledge, especially for complex questions.

2.4 Schema Linking

Schema linking is a technique that identify tables and columns mentioned in the natural language question. It can be completed by two kind of methods: string matching-based methods and neural network-based methods. UniSAr [12] and IRNet [15] find out n-grams from the question and then check which column or table they respectively match. There are two types of match that are given to scheme items: exact match (e.g. the table name ”movie” is exactly matched by ”movie”) and partial match (e.g. the column name ”movie_id” is partially matched by ”movie”). In addition, IRNet [15] also label n-grams as a column or a table. If a n-gram can match both table and column like the ”movie”, it will be recognized as a column. The scheme linking method of RAT-SQL [54] is similar to that of IRNet [15] but the match information is used in a different way. Although string match-based methods is simple and useful, it can not handle some cases like synonyms. Therefore, neural networks are used for schema linking to solve these problems. Schema-GNN [4] assesses the relevance between question words and tables/columns by the combination of cosine similarity and the output of a linear classifier. While Schema-GNN only considers local information, Global-GNN [3] uses a graph convolutional network to capture the full context. More recently, RESDSQL [23] and Codes [24] uses BERT and a multi-layer perceptron classifier to compute relevance and then remove irrelevant tables and columns. DTS-SQL [43] and MAC-SQL [55] completed the same procedure using a large language model.

2.5 SQL Correction

After scheme linking, SQL queries are generated by the model using question and schema information. However, they are not guaranteed as correct SQL queries. It is possible that a SQL query cannot be run successfully due to execution errors like syntax error or it is executable but the returned result is not expected. To address this challenge, some researches use large language model to refine SQL queries. DIN-SQL [42] directly request the model to find the errors and fix the generated SQL query, while

MAC-SQL firstly executes the SQL query to obtain the error and the exception and then input them with the wrong SQL query to the model for producing a refined answer. In addition to errors and wrong SQL query, CHESS [51] also inputs the database schema, the question and the execution result. Therefore, CHESS [51] can handle executable SQL queries that gives a wrong result. Furthermore, Self-Debugging [7] uses some (question, true SQL query) pairs as demonstrations and ask the model to explain the wrong SQL query, which can improve the accuracy of SQL query correction.

Instead of relying on large language models, UniSAr [12] creates a schema graph and find missing components in generated SQL queries through the graph.

2.6 Retrieval Architectures

The neural network-based schme linking can be thought as a retrieval task where relevant tables and columns are expected to be found. In terms of the interaction way between questions and retrieval items, the retrieval models can be divided into three categories: cross encoder, bi encoder and hybrid.

2.6.1 Cross-Encoder

The architecture of cross-encoder is shown below. It takes the concatenation of the question and the item (e.g. table or column in text-to-SQL scenario) as the input of encoder like BERT [9]. This encoding way provides full interaction between question tokens and item tokens, resulting in a effective retrieval.

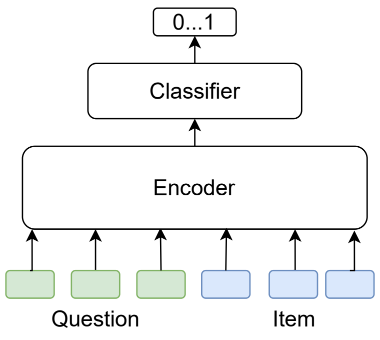


Figure 2.2: The architecture of cross-encoder where green and blue rectangles respectively are question words and item words. The output of classifier is a vector that contains values between 0 and 1.

Then, the output of encoder is feed to a classifier (e.g. a multi-layer perceptron classifier) to produce a vector that contains the probability of whether the question and the item are relevant. In text-to-SQL area, this kind of architecture is used in SLSQL [21], RESDSQL [23] and Codes [24].

2.6.2 Bi-Encoder

Bi-encoder is also called dual-encoder, where the question and the item are separately encoded. The question encoder and item encoder can either share the same weights or use different weights. Then, computing the similarity (e.g. cosine similarity) between the aggregated representation of the question and the item. The aggregated representations can be obtained from the encoder directly (e.g. the embedding of CLS token in BERT can be used to represent the whole sequence) or by averaging across token embeddings.

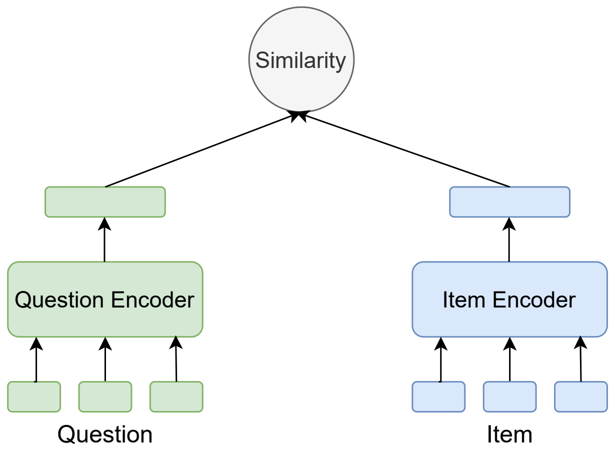


Figure 2.3: The architecture of cross-encoder where green and blue rectangles respectively are question words and item words. The output of encoders is a aggregated representation for the input.

The bi-encoder architecture does not have interaction between the question and the item during encoding, which makes it less effective than cross-encoder. However, this separate encoding allows to pre-compute the embedding for all items, which reduces a large amount of time. In text-to-SQL area, the bi-encoder architecture is used in ZeroNL2SQL [14].

2.6.3 Hybrid Architectures

More recently, some hybrid architectures are developed to combines the advantages of cross-encoder and bi-encoder. Dual-Cross-Encoder [29] jointly encode each item with pseudo-questions to produce the question-informed representation for items. Poly-Encoder [19] encodes items as in the bi-encoder but changes the encoding way for questions. It initially represents the question by several vectors and then uses these vectors and the item embedding to compute the final question embedding via attention mechanism. ERNIE-Search [40] leverages knowledge distillation [16] and a cross-encoder to improve the effectiveness of the bi-encoder.

2.7 Ranking Methods

Pointwise, pairwise and listwise are three different approaches for modelling ranking. The pointwise approach compute the relevance of a item independently of other items. Pairwise method takes a pair of items as input and predicts which item is more relevant [18], while listwise approach considers the whole list of items [58].

Chapter 3

Methodology

3.1 The architecture of schema linking

ColBERT [20] which is a hybrid retrieval architecture is selected for schema linking in the project, since it not only allows offline computation for the representations but also achieves a comparable results to the cross-encoder on other retrieval datasets. In addition, compared with other hybrid architectures, it is easier for implementation and modification.

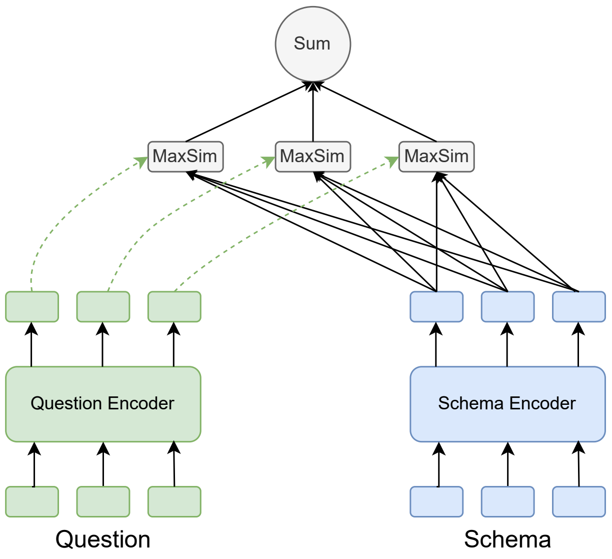


Figure 3.1: The architecture of ColBERT where green and blue rectangles respectively are question words and schema items. The relevance score is computed through late interaction between question and schema.

As shown in Figure 3.1, there are a question encoder and a schema encoder, which encodes their input into a set of token embeddings. The number of token embeddings is usually larger the length of question or schema since one word can be tokenized into two or more tokens (e.g. the tokens for the column ”movie_id” are ”movie”, ”_” and ”id”). Different from bi-encoder that uses two aggregated embeddings to compute the relevance directly, ColBERT introduces late attention mechanism. Specifically, for each token in a schema item, the similarity is calculated between it and each question token. The maximum similarity is chosen as the term score of the token. Then, the sum of token scores is the final score of the schema item.

3.2 The overview of text-to-SQL framework

The entire text-to-SQL framework is displayed in Figure 3.2, including three modules: schema linking, SQL generation and SQL correction. Firstly, natural language question from the user and corresponding database schema are feed into the schema linking model that is built on ColBERT (as described in Section 3.1). The output of schema linking model is top-k_1 relevant tables and each tables contains top-k_2 relevant columns, where k_1 and k_2 are hyperparameters that need to be defined.

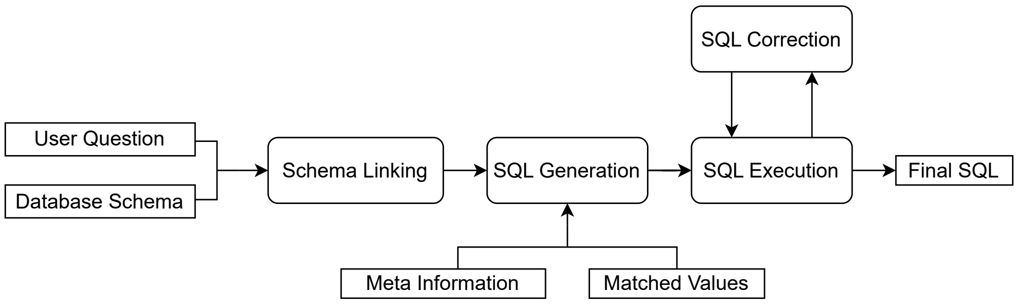


Figure 3.2: The overview of the text-to-SQL framework, which comprises three parts. (a) schema linking, which filters irrelevant tables and columns to reduce noise for SQL generation. (b) SQL generation, which takes user question, relevant schema items with their meta information and matched values as inputs and then produce SQL queries. (c) SQL correction, which fixes the SQL query that has execution errors

After schema linking, these relevant tables and columns are combined with meta information (e.g. data constraints, foreign keys) and some matched values. Matched values are recognized by searching in the database. For example, given the question ”What is the percentage of rated movies were released in year 2021?”, it can be found that 2021 is a value for the column ”movie_release_year” in the table ”movie”. Adding matched values like ”movies.movie_release_year = 2021” is useful for the generation of accurate predicates for the SQL query. Then, they are input into the SQL generation model that has a decoder-only architecture to produce SQL queries via beam search. The first SQL query which has no execution errors is selected as the final prediction. If all SQL query are not executable, the first one will be entered into the SQL correction part. A SQL query may have one or more errors while only one error can be reported per execution. Therefore, the SQL query may need to be executed and fixed many times until there is no error. The SQL correction does not use language models, instead it solves execution errors via different strategies.

Chapter 4

Implementation

4.1 Schema Linking

4.1.1 Question & Schema Encoder

The ColBERT architecture separately encodes the question and the item (see Figure 3.1). The original implementation of ColBERT [20] leverages a single BERT model among two encoders but adds a special token before [CLS] token to distinguish input sequences. However, since the frequency and distribution of words between question and schema are different, encoding is completed through two models that do not share the weight of models. In addition, following RESDSQL [23], RoBERTa [38] model which is a robustly optimized BERT is chose as the model for encoding.

4.1.1.1 Question Encoder

As mentioned in Section 2.3, external knowledge evidence is helpful for capturing the semantic meaning of the question. Therefore, the input sequence of the question encoder is the concatenation of external knowledge evidence and question: X = [Evidence] [Question]. It is firstly tokenized into tokens $q_1q_2...q_l$. The total number of tokens is a hyperparameter, which is set as 128. If the length of sequence $q_1q_2...q_l$ is longer than the total number of tokens, only the first 128 tokens will be retained. Otherwise, it is padded with special tokens to reach the length of 128. The token padding can be view as a argumentation technique. It help learn to expand question with new terms or adjust the importance of existing terms depending on their relevance to the query, which is critical for ColBERT's effectiveness [20].

After the sequence of tokens is obtained, it is feed into RoBERTa [38] to compute a contextualized representation for each token. Then, these representations are input into a linear layer and are normalized. In summary, the question encoding can be formulated as:

$$E_q = \text{Normalize}(\text{Linear}(\text{RoBERTa}(q_1 q_2 \dots q_l \#\# \dots \#))) \tag{4.1}$$

where E_q represents the final embedding of tokens, $q_1q_2...q_l$ is the sequence of tokens and # represents padded tokens.

4.1.1.2 Schema Encoder

The schema encoder is similar to the question encoder but has a different input. To capture the relationship between columns in the table and the relationship between between tables in the database, the input of the schema encoder is the whole schema that involves all tables and columns, rather than using a single table or column. In addition, some tables and columns are ambiguous (e.g. the column ”movie langs” which means the language of movie contains abbreviation in its name). To better understand the semantics of schema items, comments are incorporated with their corresponding schema items. Other meta information (e.g. data type) is not included since adding too much information may exceed the token length limitation of RoBERTa model and such information is less significant for semantic representation learning. Therefore, the input sequence of the schema encoder is the flatten schema items in default order with comments: S = $\mid t_1(*) : c_1^1(*), \cdots, c_1^{n_1}(*) \mid \cdots \mid t_N(*) : c_1^N(*), \cdots, c_{n_N}^N(*)$, where t, c, * represents table names, column names and comments respectively, vertical line and comma is used to separate tables and columns.

Similarly, this sequence passes through tokenizer, RoBERTa model, linear layer and normalization to produce the representation for each token. Since the length of schema with comments is much longer than that of question plus external knowledge evidence. The total number of tokens is defined as 512. For some extremely long schema, the tokens after 512 tokens are discarded in the training. In contrast, during the inference, a new schema is created to contains these tokens. The schema encoding can be formulated as:

$$E_s = \text{Normalize}(\text{Linear}(\text{RoBERTa}(s_1 s_2 \dots s_p \#\# \dots \#))) \tag{4.2}$$

where E_s represents the final embedding of tokens, $s_1s_2...s_p$ is the sequence of tokens and # represents padded tokens.

4.1.2 Late Attention

Once the encoding is completed, the score of each scheme item can be computed using the tokens belong to it. A column is represented by the column name and the column comment, while a table involves the table name, table comment and all column in the table. As shown in Figure 3.1, for each token in the table or column, calculating the cosine similarity between it and the question tokens and picking the maximum one as the final similarity for this token. The score of the table or column is a summation of tokens' similarity. During inference, tables in the database and columns in each table are ranked according their scores. For training, the relevance probability of schema items are computed via listwise softmax over these scores. The disadvantage of listwise approach is that a substantial computational cost is needed when the length of list is large. For example, passage retrieval task has hundreds of thousands of documents in the list. However, it is feasible in text-to-SQL task since the number of tables in a database and the number of columns in a table are less than 50. The influence of three ranking approaches (pointwise, pairwise, listwise) on the training of schema linking will be further analysed in Section 5.5.1.

4.1.3 Loss Function for Encoders

Both question and schema encoder compromises a RoBERTa model and a linear layer. The RoBERTa model is initialized by using Facebook pretrained RoBERTa-large model that has 355 million parameters and the linear layer is initialized by default. Their parameters need to be updated to fit text-to-SQL task. The loss function includes two parts: relevance prediction (RP) loss and masked language modelling (MLM) loss. The training set is imbalanced since only a few tables and columns that appear in the true SQL query are relevant (positive) and the rest of them are irrelevant (negative), which causes serious training bias. To address the imbalance problem, the focal loss [32] is selected as the relevance prediction loss, which is defined as:

$$\mathrm{FL}(y,p) = -\alpha_t(1-p_t)^\gamma \log(p_t) \tag{4.3}$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases}$$

where y is the ground truth label, p is the predicted probability, α_t is the pre-defined weight of positive/negative examples, $(1-p_t)^\gamma$ is the adjustable weight of examples based on the difficulty of prediction and γ is a pre-defined hyperparameter.

Therefore, the relevance prediction loss for tables and columns can be formulated as, respectively:

$$\mathcal{L}_{RP-Tables} = -\frac{1}{N}\sum_{i=1}^{N} FL(y_i, p_i) \tag{4.4}$$

$$\mathcal{L}_{RP-Columns} = -\frac{1}{M}\sum_{i=1}^{N}\sum_{k=1}^{n_i} FL(y_k^i, p_k^i) \tag{4.5}$$

where N is the number of tables, M is the total number of columns in the database, n_i is the number of columns in the i-th table, y_i is the ground truth label which is 1 if the i-th table is relevant and 0 otherwise, y_i^k is the ground truth label which is 1 if the k-th column in the i-th table is relevant and 0 otherwise, p_i and p_i^k is the predicted relevance probability of the i-th table and of the k-th column in the i-th table.

Furthermore, inspired by GAP [48] and GRAPPA [59], masked language modelling is employed for improving the generalization of schema linking. For tokens in both question and schema, they have 15% probability to be masked. Special tokens, including [CLS], [SEP] and [PAD], are not be masked. The multi-class cross-entropy on predicting the masked tokens is used as the MLM loss. The only difference between question MLM loss and schema MLM loss is that they have different number of masked tokens.

$$\mathcal{L}_{MLM-Question} = \mathcal{L}_{MLM-Schema} = -\frac{1}{L}\sum_{i=1}^{L}\sum_{c=1}^{C} y\log(p) \tag{4.6}$$

where y is the ground truth label that is 1 if the token at position i belongs to class c and 0 otherwise, p is the predicted probability that the token at position i belongs to class c, L is the number of masked tokens and C is the number of classes.

A simple approach for combining these four losses is to directly sum them, which gives each loss a equal weight. However, it may not produces the best result since the difficulty and importance of these tasks are different. Our model defines the weights of multiple losses as learnable parameters [31]. Thus, the loss function is formed as:

$$\begin{aligned}\mathcal{L} = f(\frac{1}{B}\sum_{i=1}^{B}(\mathcal{L}_{RP-Tables})_i) + f(\frac{1}{B}\sum_{i=1}^{B}(\mathcal{L}_{RP-Columns})_i) \\ + f(\frac{1}{B}\sum_{i=1}^{B}(\mathcal{L}_{MLM-Question})_i) + f(\frac{1}{B}\sum_{i=1}^{B}(\mathcal{L}_{MLM-Schema})_i)\end{aligned} \tag{4.7}$$

$$f(L_\tau) = \frac{1}{2\cdot c_\tau^2}\cdot L_\tau + ln(1 + c_\tau^2) \tag{4.8}$$

where B is the number of batch size, c_τ is the learnable parameter which is initialized as 1 and $ln(1 + c_\tau^2)$ is the regularization that avoids the value of parameters become too large.

4.1.4 Training details

The schema linking model is trained using full fine tuning with AdamW [39] optimizer. The learning rate is set to 1e-5 and it is adjusted via linear warm-up (the first 2% training steps) and cosine decay. The size of batch is 4 and gradients are accumulated over 8 steps before performing an update. For the hyperparameters of focal loss, the γ in $(1 - p_t)^\gamma$ is set to 2 and the weight α is 0.75 for positive examples and is 0.25 for negative examples. Moreover, the early stop is employed to avoid overfitting, whose steps is defined to 8. The training is completed on a server with one NVIDIA A40 (48G) GPU, one Intel(R) Xeon(R) Platinum 8358P CPU, 80 GB memory and Ubuntu 20.04 operating system and it takes about 48 hours.

4.2 Matched Values Identification

The implementation of match values identification follows Codes [24], which is a two-stage retrieval. Firstly, BM25 indexes are created for all values in the database, allowing to obtain hundreds of potentially relevant values for the question. In the second stage, the longest common substring (LCS) algorithm is utilized to further find the most relevant values.

4.3 SQL Generation

Codes [24] is used for SQL generation since it is a state-of-the-art small-size text-to-SQL model. It has a decoder-only architecture. The beam search is employed in the generation and the number of beams is 4, which means the model predicts 4 SQL queries for each sample. The first SQL query that has no execution errors is chose as the final outcome.

4.4 SQL Correction

When all predictions are not executable, the first SQL query is input into the SQL correction part. Execution errors can be divided into six categories:

- No such column: this error happens when a predicted column in SELECT, WHERE or JOIN ON clause are not in the database. To address it, replacing the non-existing column by the most similar column in the same table. Firstly, the table of the wrong column need to be found. For some cases, it is directly written in the execution error (e.g. no such column: schools.opened, where ”schools” is the table). If the table is not reported, it will be identified by searching FROM keyword which is the most closest to the non-existing column. The table is on the right of this FROM keyword. Then, the fasttext model [5] which is trained with subword infomation is used to encode the non-existing column and all columns in the table and cosine similarity is computed between them to find the most similar column. Compared with BERT [9] and Word2Vec [41], the fasttext model does not significant increases the inference time and it can handle out-of-vocabulary words (e.g. schools.admflname1).

- Syntax error: If a predicted table or column is the same as the reserved keywords such as ORDER and SET, the syntax error will be caused. It can be fixed by wrapping the table or column in quotes to distinguish from the keywords. For instance, order becomes ’order’.
- Misuse of aggregate: this error happens when a aggregate operation like sum() is used incorrectly. Therefore, the solution of solving the error is to remove the wrong aggregate operation from the SQL query.
- Ambiguous column name: it happens when the table and column have identical name (e.g. colour.colour). The solution of ”no such column” error can be applied to solve it.
- No such table: this error occurs when a predicted table is non-existing. To fix it, replacing the table with the most similar table in the database using the fasttext model and cosine similarity.
- No such function: generating functions (e.g. datediff) which are not supported by SQL causes this error. These wrong functions are deleted to solve the error.

Chapter 5

Evaluation

5.1 Dataset

Previous benchmarks like Spider [60] and WikiSQL [63] only provide a small number of values for the tables, which are different from the real-world scenarios. Therefore, BIRD [26] is selected as the dataset, which not only provide large scale databases but also collect external knowledge evidence for examples. It has 95 databases with a total size of 33.4 GB, covering 37 professional domains such as healthcare and sports. The training set of BIRD contains 9428 examples, which is used to train the schema linking model. Since the test set is not released, all evaluations are conducted on the development set that involves 1534 examples. The databases of the development set are not in the training set.

5.2 Evaluation Metrics

The evaluation can be divided into two parts: evaluating the schema linking model only and evaluating the whole text-to-SQL pipeline which comprises schema linking, SQL generation and SQL correction. Each part employs distinct evaluation metrics.

To solely evaluate the scheme linking model in aspects of effectiveness and efficiency, AUC metric and average processing time are used. The full name of AUC is Area Under the Curve where the curve means ROC (Receiver Operating Characteristic) curve that plots the true positive rate against the false positive rate. AUC metric quantifies the overall ability of the model to discriminate between positive and negative classes. Its value ranges from 0 to 1, where 1 represents perfect classification, 0.5 means the ability of the model is equivalent to random guessing. Furthermore, the model' efficiency is assessed by dividing the total consumed time of schema linking by the number of examples.

For the evaluation on the whole text-to-SQL pipeline, there are also two metrics: execution accuracy (EX) and valid efficiency score (VES). The definition of execution accuracy is the percentage of examples whose predicted SQL query can return the same results as the ground truth SQL query after execution [44], which is expressed as:

$$\text{EX} = \frac{\sum_{n=1}^{N} I(V_n, \hat{V}_n)}{N} \tag{5.1}$$

$$I(V, \hat{V}) = \begin{cases} 1 & V = \hat{V}, \\ 0 & V \neq \hat{V}. \end{cases}$$

where N is the total number of examples in the evaluation set, the results V_n are returned by the n-th ground-truth SQL and the results $\hat{V}_n$ are return by the predicted SQL query, $I(V, \hat{V})$ is an indicator function that returns 1 if two results are identical and 0 otherwise.

Valid efficiency score is define as the average execution efficiency of valid predicted SQL queries whose execution results is identical to that of ground truth SQL query [26]. It is a comprehensive evaluation metric since it takes both efficiency and accuracy into account. The formulation of VES is shown as:

$$\text{VES} = \frac{\sum_{n=1}^{N} I(V_n, \hat{V}_n) \cdot R(Y_n, \hat{Y}_n)}{N}, \tag{5.2}$$

$$R(Y_n, \hat{Y}_n) = \sqrt{\frac{E(Y_n)}{E(\hat{Y}_n)}}$$

where N is the total number of examples in the evaluation set, V_n are returned by the n-th ground-truth SQL Y_n and the results $\hat{V}_n$ are return by the predicted SQL query $\hat{Y}_n$, $I(V, \hat{V})$ is an indicator function as mentioned above, $R(Y, \hat{Y})$ means relative execution efficiency of the predicted SQL query compared with the ground truth SQL query, $E(\cdot)$ is a function to measure the absolute execution efficiency (namely, running time) for each SQL query, which uses square root function to minimize random abnormal errors.

5.3 Environments

All evaluations are conducted using PyTorch 1.13.1 on the same server which is used for the training of scheme linking model: one NVIDIA A40 (48G) GPU, one Intel(R) Xeon(R) Platinum 8358P CPU, 80 GB memory and Ubuntu 20.04 operating system. The whole text-to-SQL pipeline takes around 40 minutes.

5.4 Baselines

The baseline for the evaluation on schema linking is a cross-encoder based schema linking model which is used for Codes [24]. For evaluating the entire text-to-SQL, combining this cross-encoder based model with the Codes model (SQL generation model) as the baseline. Codes models of different sizes which are fine tuned on the training set of BIRD with external knowledge can be downloaded from Hugging face. Codes-1B and Codes-3B are employed in the evaluation of text-to-SQL.

5.5 Evaluation on Scheme Linking

5.5.1 The influence of ranking methods

This section investigates how pointwise, pairwise and listwise methods affects the effectiveness of ColBERT based schema linking. The output of ColBERT using pointwise ranking is not a sum of maximum similarity. Instead, a mean of maximum similarity is used, which can directly represent the relevance probability.

	pointwise	pairwise	listwise
Table AUC	0.834	0.945	**0.964**
Column AUC	0.789	0.928	**0.981**
Total AUC	1.623	1.873	**1.945**

Table 5.1: The performance of ColBERT models with three different ranking methods.

As presented in Table 5.1, using pairwise or listwise approach can achieve higher table AUC score and column AUC score compared with pointwise method, which means comparison between schema items is necessary for ColBERT-based schema linking. Furthermore, it is found that the difference between table AUC and column AUC is reversed when changing from pairwise to listwise. The number of columns in a table is much more than the number of tables in the database. Therefore, listwise approach is more beneficial for columns.

5.5.2 The comparison with the baseline

The baseline is a cross-encoder based model which uses pointwise method. Table 5.2 shows that the overall performance of ColBERT model is slightly superior to that of baseline due to the increase in the column AUC metric which is caused by incorporating the comparison between columns. However, the baseline achieves a higher AUC score on table relevance classification compared with ColBERT model. The reason is that a long schema that has more than 512 tokens is split into two schema, which is detrimental to represent some tables correctly.

	Baseline	ColBERT
Table AUC	**0.976**	0.964
Column AUC	0.957	**0.981**
Total AUC	1.933	**1.945**

Table 5.2: The effectiveness evaluation of baseline and ColBERT model in table, column and total AUC scores. The best results are emphasized in bold.

In addition, the average time of schema linking for examples is computed using two models. The ColBERT model is approximately 30% faster than the baseline since it allows the pre-computation of embedding of schema items.

	Baseline	ColBERT
Average Time	0.136	0.0965

Table 5.3: The average time for completing scheme linking on the development set (1534 examples) using baseline and ColBERT model, respectively.

5.6 Evaluation on Text-to-SQL

To verify the effectiveness of ColBERT-base schema linking, using it with two Codes models which are different in parameter size to generate SQL queries.

Approach	EX(%)	VES(%)
Baseline (Cross-Encoder + Codes-1B)	47.26	50.03
ColBERT + Codes-1B	49.41	51.8
ColBERT + Codes-1B + SQL Correction	**49.67**	**52.16**

Table 5.4: The evaluation results of text-to-SQL using Codes-1B on the development set of BIRD with external knowledge. The best performance is emphasized in bold.

Since 99% questions has less than 4 relevant tables and 99% tables includes less than 4 relevant columns, the number of retained tables and columns is set to 3 to reduce the interference of useless information. As shown in Table 5.4, combining ColBERT base schema linking with Codes-1B slightly outperforms the baseline, gaining the increase of 2.15% in execution accuracy (from 47.26% to 49.41%). However, the improvement of incorporating SQL correction is very small, which is only 0.26% in execution accuracy. The rising trend in valid efficiency score (VES) aligns with the trend in execution accuracy(EX). Larger improvement in EX corresponds to greater growth in VES, following the definition of VES.

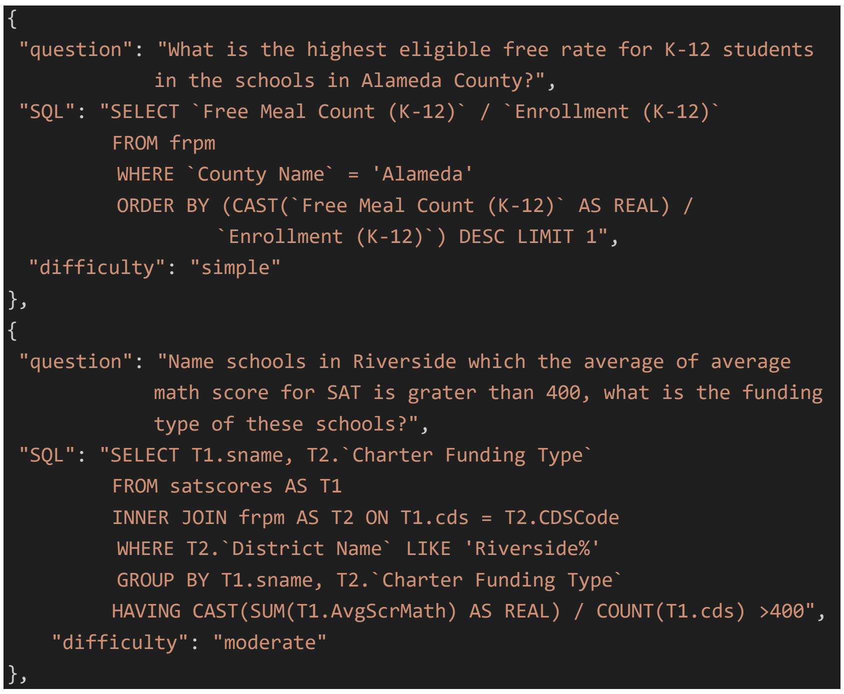
Furthermore, the same evaluation using a larger SQL generation model (Codes-3B) is conducted and the results are presented in Table 5.5. Compared with Codes-1B, the EX and VES of baseline increases from 47.26% to 50.72%and from 50.03% to 56.32%, which indicates that SQL queries generated from Codes-3B are more correct and efficient. Combining ColBERT based schema linking with Codes-3B just achieves a improvement of 0.39% in EX (from 50.72% to 51.11%), which is lower than that of using ColBERT based schema linking with Codes-1B since the larger model has a more powerful capability. Adding SQL correction for Codes-3B lead to a growth of 0.32% in EX, which is similar to using it with Codes-1B.

Approach	EX(%)	VES(%)
Baseline (Cross-Encoder + Codes-3B)	50.72	56.32
ColBERT + Codes-3B	51.11	56.24
ColBERT + Codes-3B + SQL Correction	**51.43**	**56.5**

Table 5.5: The evaluation results of text-to-SQL using Codes-3B on the development set of BIRD with external knowledge. The best performance is emphasized in bold.

A notable point in Table 5.5 is that the value of VES decreases from 56.32% to 56.24% when using ColBERT base schema linking without SQL correction, even if there exists a growth in EX. To investigate the reason, the examples in the evaluation set are categorized into three classes (simple, moderate, challenging) based on the difficulty of question and valid efficiency score (VES) is computed for each class.

Figure 5.1 demonstrates a simple example and a moderate example. It can be observed that the length of question and of SQL query increase as the difficulty level become higher. Additionally, the SQL query may involve more complex operations such as ”GROUP BY” and ”HAVING”.



```
{
 "question": "What is the highest eligible free rate for K-12 students
             in the schools in Alameda County?",
 "SQL": "SELECT `Free Meal Count (K-12)` / `Enrollment (K-12)`
         FROM frpm
         WHERE `County Name` = 'Alameda'
         ORDER BY (CAST(`Free Meal Count (K-12)` AS REAL) /
                  `Enrollment (K-12)`) DESC LIMIT 1",
  "difficulty": "simple"
},
{
 "question": "Name schools in Riverside which the average of average
             math score for SAT is grater than 400, what is the funding
             type of these schools?",
 "SQL": "SELECT T1.sname, T2.`Charter Funding Type`
         FROM satscores AS T1
         INNER JOIN frpm AS T2 ON T1.cds = T2.CDSCode
         WHERE T2.`District Name` LIKE 'Riverside%'
         GROUP BY T1.sname, T2.`Charter Funding Type`
         HAVING CAST(SUM(T1.AvgScrMath) AS REAL) / COUNT(T1.cds) >400",
    "difficulty": "moderate"
},
```

Figure 5.1: The illustration of examples with different levels of difficulty, including a simple example and a moderate example. The challenging example is not contained since the question and SQL query are too long.

As shown in Table 5.6, there are 925 simple examples, 465 moderate examples and 144 challenging examples in the development set of BIRD. While VES of moderate and challenging examples are enhanced, it can be found that VES of simple examples decreases from 65.97% to 60.24%, which is the reason of the overall VES reduction (from 56.32% to 56.24%) in Table 5.5.

	Simple	Moderate	Challenging
Count	925	465	144
Baseline (Cross-Encoder + Codes-3B)	**65.97**	44.58	32.2
ColBERT + Codes-3B	60.24	**52.56**	**42.49**

Table 5.6: The valid efficiency scores (VES) for examples of different difficulty. The first row represent the number of simple, moderate and challenging examples.

There are two potential reasons for the decreases in the VES of simple examples. The first one is that valid SQL queries that can return expected data become fewer and the second reason is that generated SQL queries take more time for execution. By measuring the execution accuracy (EX) of simple examples (Baseline: 58.81%, ColBERT + Codes-3B: 60%), a improvement can be found. Therefore, the first reason can be excluded from consideration.

Generated SQL queries for simple examples becomes less efficient, probably because overall length of SQL queries becomes longer to satisfy the requirement of moderate and challenging examples (as shown in Figure 5.1).

Furthermore, execution accuracy (EX) examples in three difficulty levels are computed to check whether SQL correction is useful for complex examples. Table 5.7 presents SQL correction can improve EX of simple, moderate and challenging examples when using Codes-3B, while it only achieves higher EX for simple examples when using Codes-1B and EX of moderate and challenging examples remain unchanged.

	Simple	Moderate	Challenging
ColBERT + Codes-1B	57.95	38.28	30.56
ColBERT + Codes-1B + SQL correction	**58.38**	38.28	30.56
ColBERT + Codes-3B	60	38.49	34.72
ColBERT + Codes-3B + SQL correction	**60.22**	**38.92**	**35.42**

Table 5.7: The execution accuracy (EX) for examples of different difficulty. The first two columns are produced using Codes-1B without or with SQL correction and the last two columns are produced using Codes-3B without or with SQL correction.

In summary, both ColBERT model and SQL correction can enhance the performance of text-to-SQL. The ColBERT model provides a more significant improvement for the small SQL generation model (e.g. Codes-1B) compared to a larger one (Codes-3B). The improvement of SQL correction is consistent but slight, which indicates that execution errors are not the main reason for faulty SQL queries that cannot produce correct answers, especially for wrong SQL queries generated from small SQL generation model for non-simple questions.

5.7 Error Analysis

Execution errors caused by Codes-1B and Codes-3B are further analysed. For Codes-1B, all predicted SQL queries have execution errors for 75 examples, while this number is 72 for Codes-3B. As presented in Table 5.8, the most frequent error is ”no such column”, while ”no such table” rarely occurs. This also indicates that the difficulty of predicting columns is more then that of predicting tables since the number of columns is much more than the number of tables. The number of occurrences of ”syntax error” is significantly less compared to ”no such column” error but still more than other errors. This kind of error is dependent of whether the used databases involves table or column names which is similar to SQL keywords. For instance, the databases of the development set of BIRD have the table names ”order” and ”sets”. Therefore, the predicted SQL queries contain table names ”order” and ”set”, which causes conflicts with SQL keywords ”ORDER” and ”SET”, respectively.

	Codes-1B	Codes-3B
No such column	77	73
Syntax error	7	10
Misuse of aggregate	3	1
Ambiguous column name	2	2
No such table	2	3
No such function	1	2

Table 5.8: The execution errors collected from SQL queries that are input to SQL correction. One wrong SQL query contains one or more errors. These errors are divided into six classes.

Chapter 6

Conclusions

6.1 Summary

In conclusion, this project enhances the performance of open-source small-size text-to-SQL models by employing the ColBERT-based schema linking and SQL error correction. The ColBERT model separately encodes the question and the database schema and uses late attention mechanism to compute the score for tables and columns which is a summation of maximum similarity between tokens. Then, these scores are used to sort tables and columns to recognize the most relevant tables and the most relevant columns in these tables, which will be combined with database meta information and matched values to help the generation of SQL queries. The ColBERT-based schema linking can improve the performance of SQL generation models in different size but the increases of small models is higher than that of larger models. In addition, this method can significantly reduce the time for schema linking since it allows to compute the embedding for schema offline. For training the ColBERT model, there are three approaches can be utilized and listwise approach is feasible and the most effective.

Furthermore, some generated SQL queries are not executable due to execution errors. Theses errors can be divided into six categories and ”no such column” error is the majority of all errors. The SQL correction is developed to fix execution errors in the generated SQL queries using different strategies. However, only a small number of fixed SQL queries can return expected data from the database, which means that execution errors are not the main problem for faulty SQL queries especially for those of them generated by small models (e.g. Codes-1B) for moderate and challenging questions.

6.2 Futhre Work

A unsolved problem in this project is the extremely long schema is split into two sub-schema which breaks the representation of some tables and therefore causes the decrease in the table AUC metric. The suggestion for solving this problem in the future is to treat the schema linking as a two-stage process for the long schema, which firstly removes irrelevant tables to reduce the length of schema and then ranks the rest of tables and columns in the second stage. Additionally, there is a decrease in execution accuracy (EX) of simple examples when applying ColBERT-based schema linking with Codes-3B. Fine tuning Codes-3B with trained ColBERT model may help address this issue. Lastly, only Codes models are used in the evaluation. To further verify the effectiveness of ColBERT-based schema linking and SQL correction, combining them with other models is useful.

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[3] Ben Bogin, Matt Gardner, and Jonathan Berant. Global reasoning over database structures for text-to-sql parsing. *arXiv preprint arXiv:1908.11214*, 2019.

[4] Ben Bogin, Matt Gardner, and Jonathan Berant. Representing schema structure with graph neural networks for text-to-sql parsing. *arXiv preprint arXiv:1905.06241*, 2019.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[6] Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems*, 34:7664–7676, 2021.

[7] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*, 2023.

[11] Longxu Dou, Yan Gao, Xuqi Liu, Mingyang Pan, Dingzirui Wang, Wanxiang Che, Dechen Zhan, Min-Yen Kan, and Jian-Guang Lou. Towards knowledge-intensive text-to-sql semantic parsing with formulaic knowledge. *arXiv preprint arXiv:2301.01067*, 2023.

[12] Longxu Dou, Yan Gao, Mingyang Pan, Dingzirui Wang, Wanxiang Che, Dechen Zhan, and Jian-Guang Lou. Unisar: A unified structure-aware autoregressive language model for text-to-sql. *arXiv preprint arXiv:2203.07781*, 2022.

[13] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*, 2023.

[14] Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, and Xiaoyong Du. Interleaving pre-trained language models and large language models for zero-shot nl2sql generation. *arXiv preprint arXiv:2306.08891*, 2023.

[15] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[17] Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. Knowledge-to-sql: Enhancing sql generation with data expert llm. *arXiv preprint arXiv:2402.11517*, 2024.

[18] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD*

International Conference on Knowledge Discovery & Data Mining, pages 2553–2561, 2020.

[19] Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. *arXiv preprint arXiv:1905.01969*, 2019.

[20] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[21] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. Re-examining the role of schema linking in text-to-sql. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954, 2020.

[22] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[23] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075, 2023.

[24] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *arXiv preprint arXiv:2402.16347*, 2024.

[25] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13076–13084, 2023.

[26] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024.

[27] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[28] Yuntao Li, Zhenpeng Su, Yutian Li, Hanchu Zhang, Sirui Wang, Wei Wu, and Yan Zhang. T5-sr: A unified seq-to-seq decoding strategy for semantic parsing. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[29] Zehan Li, Nan Yang, Liang Wang, and Furu Wei. Learning diverse document representations with deep query interactions for dense retrieval. *arXiv preprint arXiv:2208.04232*, 2022.

[30] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, et al. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *arXiv preprint arXiv:2403.09732*, 2024.

[31] Lukas Liebel and Marco Körner. Auxiliary tasks in multi-task learning. *arXiv preprint arXiv:1805.06334*, 2018.

[32] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[33] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *arXiv preprint arXiv:2012.12627*, 2020.

[34] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*, 2023.

[35] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[36] Qian Liu, Dejian Yang, Jiahui Zhang, Jiaqi Guo, Bin Zhou, and Jian-Guang Lou. Awakening latent grounding from pretrained language models for semantic parsing. *arXiv preprint arXiv:2109.10540*, 2021.

[37] Xiping Liu and Zhao Tan. Divide and prompt: Chain of thought prompting for text-to-sql. *arXiv preprint arXiv:2304.11556*, 2023.

[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[39] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[40] Yuxiang Lu, Yiding Liu, Jiaxiang Liu, Yunsheng Shi, Zhengjie Huang, Shikun Feng Yu Sun, Hao Tian, Hua Wu, Shuaiqiang Wang, Dawei Yin, et al. Ernie-search: Bridging cross-encoder with dual-encoder via self on-the-fly distillation for dense passage retrieval. *arXiv preprint arXiv:2205.09153*, 2022.

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[42] Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36, 2024.

[43] Mohammadreza Pourreza and Davood Rafiei. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*, 2024.

[44] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*, 2022.

[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[46] Daking Rai, Bailin Wang, Yilun Zhou, and Ziyu Yao. Improving generalization in language model-based text-to-sql semantic parsing: Two simple semantic boundary-based techniques. *arXiv preprint arXiv:2305.17378*, 2023.

[47] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*, 2021.

[48] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13806–13814, 2021.

[49] Richard Shin, Christopher H Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. Constrained language models yield few-shot semantic parsers. *arXiv preprint arXiv:2104.08768*, 2021.

[50] Ruoxi Sun, Sercan Ö Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, et al. Sql-palm: Improved large language model adaptation for text-to-sql (extended). *arXiv preprint arXiv:2306.00739*, 2023.

[51] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

[52] Yasufumi Taniguchi, Hiroki Nakayama, Kubo Takahiro, and Jun Suzuki. An investigation between schema linking and text-to-sql performance. *arXiv preprint arXiv:2102.01847*, 2021.

[53] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[54] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*, 2019.

[55] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*, 2023.

[56] Ping Wang, Tian Shi, and Chandan K Reddy. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361, 2020.

[57] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[58] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM conference on recommender systems*, pages 269–277, 2019.

[59] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. Grappa: Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845*, 2020.

[60] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[61] Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. *arXiv preprint arXiv:2401.10506*, 2024.

[62] Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. Editing-based sql query generation for cross-domain context-dependent questions. *arXiv preprint arXiv:1909.00786*, 2019.

[63] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

Appendix A

First appendix