Preprints.org

Article

Not peer-reviewed version

FGCSQL: A Three-Stage Pipeline for Large Language Model-driven Chinese Text-to-SQL

Guanyu Jiang , Weibin Li [*] , Chenglong Yu , Zixuan Zhu , Wei LI

Posted Date: 14 February 2025

doi: 10.20944/preprints202502.1059.v1

Keywords: Text-to-SQL; LLM; CSpider; Natural language processing

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

FGCSQL: A Three-Stage Pipeline for Large Language Model-Driven Chinese Text-to-SQL

Guanyu Jiang [1], Weibin Li [2,*], Chenglong Yu [2], Zixuan Zhu [1] and Wei Li [1]

[1] Hangzhou Institute of Technology, Xidian University, Hangzhou 311231 China
[2] School of Artificial Intelligence, Xidian University, Xi'an 710071 China
[*] Correspondence: weibinli@xidian.edu.cn

Abstract: Recent advances in large language models have driven major breakthroughs in Text-to-SQL tasks. However, many challenges hinder the use of SQL parsers for cross-language tasks. In this article, we introduce FGCSQL, a novel three-stage pipeline framework to deal with three challenges: cross-language schema linking, SQL parsing potential of LLM, and error propagation in SQL parsers, in which the framework uniquely incorporates a Filtering Encoder to eliminate irrelevant database schema items, harnessing a pre-trained Generative Large Language Model fine-tuned on a carefully structured dataset for enhanced SQL parsing. Finally, a Correcting Decoder addresses error propagation, culminating in a robust system for semantic parsing tasks. Tested on the CSpider dataset, the FGCSQL showcases a substantial improvement in Exact-set-Match(EM) accuracy and EXecution accuracy(EX) metrics, validating the pipeline's architecture's effectiveness in mitigating the challenges typically confronted in Text-to-SQL conversion, especially in cross-lingual contexts. FGCSQL outstrips existing methods in execution precision, indicating the validity of our proposed method.

Keywords: Text-to-SQL; LLM; CSpider; natural language processing

1. Introduction

Because of its powerful data organization and management ability, as well as mature transaction processing and data consistency guarantee, relational database is widely used in many industries, such as finance, retail, medical and so on. However, there is a high threshold for the use of structured query language SQL, so researchers have proposed Text-to-SQL tasks that aim to translate natural language problems into structured query statements. By constructing the natural language interface of the database, this task has become a research hotspot in the natural language processing and database community, so that users can extract key information efficiently without professional background.

The challenge of Text-to-SQL lies in bridging the semantic gap between natural language and SQL queries, encoding complex natural language inputs, and decoding them into richly structured SQL query statements. Previous studies have been dedicated to addressing these challenges, treating Text-to-SQL as a sequence-to-sequence task, focusing on encoder-decoder architectures, and training machine learning models using Text-to-SQL corpora [1–4].

The process of implementing Text-to-SQL based on the encoder-decoder architecture begins with data preprocessing, where training data is collected and cleaned. Afterward, the encoder transforms a natural language query into encoded information, capturing both syntax and semantic details. The decoder then utilizes this encoded information to progressively generate the corresponding SQL query, including components such as SELECT, FROM, and WHERE.

Cross-language schema linking for Chinese Text-to-SQL needs to be strengthened.

Existing solutions exhibit significant limitations. On one hand, current approaches tend to suit simple queries, but their performance drops as the number of database schema complexities and multi-table joins increase. On the other hand, while database schemas are usually in English, user idioms(take Chinese for example) and the data they record may be in other languages. In such cases,

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

SQL parsers trained on English Text-to-SQL corpora greatly diminish in performance, or may even fail to comprehend the natural language queries correctly.

Thus, the associated benchmarks have become increasingly intricate, from single-domain datasets to multi-domain datasets, accumulating more and more complex Query-SQL pairs. Single-domain datasets collect Query-SQL pairs for a single database in real-world tasks, such as ATIS [5], SEDE [6], etc. Cross-domain datasets contain multiple databases, to test the performance of models on complex, unseen SQL queries and their ability to generalize to new domains, such as WikiSQL [7], Spider [8].

Many existing studies are conducted on Spider for its challenging nature, which involves various complex operators. Research on Chinese Text-to-SQL tasks, in comparison, is less frequent. Due to the increasing complexity of the existing test benchmarks, and the obvious difference between the description of Chinese natural language questions and the language used by the database schema items, it is necessary to strengthen cross-language schema linking for the complex Chinese Text-to-SQL test benchmarks.

The potential of open-source LLMs has not been fully tapped. Recently, open-source large language models (LLMs) have flourished, offering rich linguistic knowledge learned from vast corpora through pretraining, better capturing the diversity and variability of natural language, and understanding contextual relationships within, excelling in tasks such as programming, logical reasoning, and text generation.

Cutting-edge research combining pretrained large-scale language models with prompt engineering has achieved remarkable results [9–13].

However, most studies based on LLMs utilize closed-source ones with vast parameter scales, such as OpenAI LLMs, overlooking the potential of open-source LLMs. Addressing Text-to-SQL tasks with closed-source LLMs offers little room for optimization and often requires considerable inference costs. Although open-source LLMs are somewhat limited in context understanding and text generation quality compared to closed-source ones, this issue can be rectified through supervised fine-tuning.

Multi-stage pipeline method leads to error propagation. For Text-to-SQL tasks, end-to-end methods often increase the learning difficulty. Most existing methods adopt a multi-stage model, assisting the SQL decoder with auxiliary tasks such as schema item classification and encoding token types, collaboratively resolving the Text-to-SQL task [3,14,15].

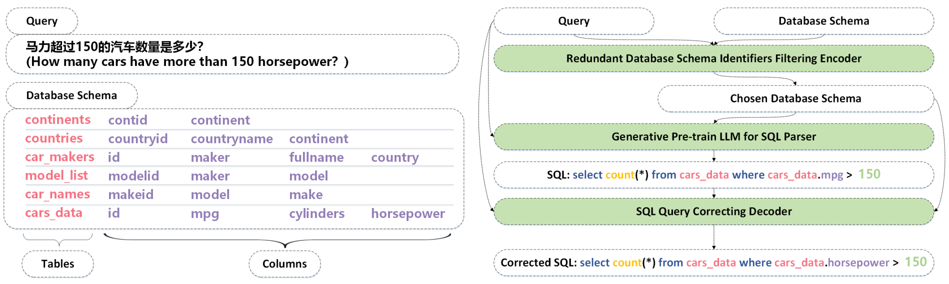
Multi-stage methods decompose complex tasks into stages, each fulfilled by specialized components or modules, thereby enhancing overall system performance. However, multi-stage pipeline methods also present the challenge of error propagation [16–18]. Error propagation refers to mistakes in one stage of the pipeline potentially cascading to subsequent stages, causing errors in system output.

Addressing error propagation is a significant challenge in designing multi-stage pipeline systems.

To explore the performance of open-source large language models on Chinese Text-to-SQL tasks, and research potential methods to strengthen cross-linguistic schema linking and reduce the learning difficulty of Text-to-SQL conversion, we conducted preliminary experiments on the Chinese Text-to-SQL CSpider dataset. Initially, we supervised fine-tuned a relatively small-scale open-source LLM such as LlaMa2-7b-chat using only basic question representations to organize training data, resulting in nearly a 40% improvement in Exact Match on the development set. This pre-experiment convinced us that using LLMs to tackle Chinese Text-to-SQL tasks is an effective future method.

We propose a three-stage pipeline method: Redundant Database Schema Items Filtering Encoder(RDsiF-Encoder for short), Generative Pre-trained LLM for SQL Parser, SQL Query Correcting Decoder, to validate the feasibility of our ideas. First, train a filtering encoder in the initial stage, which filters out database information irrelevant to the Query, enhancing the cross-lingual schema linking accuracy and reducing the learning difficulty for the SQL parser; then, in the second stage, use the supervised fine-tuned LLM to parse the Query and convert it into an SQL statement; The final stage involves training a correcting decoder to rectify the SQL statements output by the second stage, further boosting model performance.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1


Query
马力超过150的汽车数量是多少?
(How many cars have more than 150 horsepower?)
Database Schema
continents contid continent
countries countryid countryname continent
car_makers id maker fullname country
model_list modelid maker model
car_names makeid model make
cars_data id mpg cylinders horsepower
Tables
Columns
Query
Database Schema
Redundant Database Schema Identifiers Filtering Encoder
Chosen Database Schema
Generative Pre-train LLM for SQL Parser
SQL Query Correcting Decoder

```
SQL: select count(*) from cars_data where cars_data.mpg > 150
Corrected SQL: select count(*) from cars_data where cars_data.horsepower > 150
```

Figure 1. Schematic of the FGCSQL method.

Contributions

We propose a three-stage pipeline method to solve the Chinese Text-to-SQL task: Filtering redundant information - Generating SQL queries - Correcting SQL statements. Specifically, we devised an encoder that filters out redundant schema items from specific natural language queries, diminishing the interference on generative models. To bridge the gap between natural language and SQL statements, we use NatSQL as an intermediate representation, combined with supervised fine-tuning of a pre-trained large language model, improving Text-to-SQL accuracy. We collected a correcting dataset and trained a correction model to amend the output of the large language model, thereby reducing the impact of error propagation produced by the multi-stage pipeline.

We propose the Chinese information injected layer and data type-aware attention to enhance the filter encoder's understanding of Chinese semantics and data type perception, eliminate the ambiguity problem in cross-language schema linking, and avoid the interference of schema items with mismatched data types on the filter encoder.

We propose a question representation method necessary for LLM supervised fine-tuning for the Text-to-SQL task: IECQN question representation, Instruction (I), Example (E), Chosen Database schema (C), Question (Q), NatSQL (N).

We conducted a series of evaluations and analyses, and the results show that our methods have achieved state-of-the-art performance in execution accuracy on the CSpider dataset.

Outline The remainder of this paper is structured as follows. The Section 2 introduces the latest Text-to-SQL solutions and related studies. The Section 3 presents our contributions to the task. Sections 4, 5, 6 includes the expermental setup, results, and the conclusions of this work.

2. Related Work

At present, researchers in the Text-to-SQL domain have developed a variety of approaches, which can mainly be categorized into three types: encoding approaches, decoding approaches and methods based on prompt engineering. This chapter will review and discuss the latest advancements in this field.

2.1. Encoding Approaches

Many existing studies are dedicated to employing various methods to represent the semantics of questions, the structure of database schemas, as well as schema linking.

[19]proposes a weakly-supervised pretraining framework named Structure-Grounded (STRUG) to learn text-table alignment more effectively. [3] introduces a ranking-enhanced encoding for Text-to-SQL that aims to reinforce schema linking and has shown promising performance and robustness on Spider and its three robust variations.

Given that database schemas contain rich structural information, graph-based methods have been used to better encode such structures. [20]discusses the integration of relational structures into a heterogeneous graph for Chinese NL2SQL parsers, which aims to solve issues like column name reuse, natural language query descriptions, and inconsistent data presentation in databases. [21]utilizes a Graph Convolutional Network (GCN) to capture the database structure and employed a gated GCN to select relevant database information for SQL generation. [4] encodes more relationships for database schemas, such as "two columns from the same table".

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

2.2. *Decoding Approaches*

Several decoder-based approaches have been developed to reduce the complexity of SQL parsing and bridge the gap between natural language and SQL.

[22] adopts a tree-based decoding method specific to SQL syntax, recursively invoking modules to predict different SQL components. [23] recursively generates SELECT statements for SQL queries with complex nested structures, performing sketch-based slot filling for each SELECT statement. [24]proposes SemQL as an intermediate representation for SQL. Similarly, [25]introduces NatSQL as an intermediate representation to simplify SQL structure, enhancing the executability of the SQL generated by existing models.

[26] and [27] impose constraints on the decoder to prevent the generation of invalid tokens. [28] and [29] utilize an execution-guided decoding mechanism, excluding parts of SQL queries that are not executable from the output candidates.

2.3. *Prompt Engineering Approaches*

With the rise of pre-trained language models, approaches based on Large Language Models (LLMs) have significantly enhanced the performance of various tasks in the NLP field, not excluding the Text-to-SQL task. In the application of Text-to-SQL, these pre-trained models have significantly improved the depth of understanding of natural language queries and the ability to accurately match these queries with database structures.

[12] investigates how to break down the complex Text-to-SQL task into smaller sub-tasks and demonstrated how this decomposition significantly improves the performance of Large Language Models (LLMs) during inference. [12] enhances their performance by decomposing the generation of SQL queries and feeding these sub-problems' solutions into LLMs. [13] integrates previous experimental outcomes and presents a new comprehensive solution, setting a new record on the Spider leaderboard with an execution accuracy of 86.6%.

3. FGCSQL

3.1. *Redundant Database Schema Items Filtering Encoder*

In a given database D, with tables list $T = \{t_1, t_2, ..., t_{N_t}\}$ and columns lists $C = \{C_1, C_2, ..., C_{N_t}\}$, where $C_i = \{c_1, c_2, ..., c_{n_i}\}, i \in \{1, 2, ..., N_t\}$, natural language Queries typically do not involve all database schema items. Superfluous schema items are likely to negatively affect Text-to-SQL parser performance. Moreover, aligning entities mentioned in natural language Queries with database schema items is one of the challenges in Text-to-SQL tasks. Thus, we did not use all database schema items as prompts for the LLM but trained a redundant database schema items filtering encoder (RDSiF Encoder for short) instead. We define the filtering task as a classification process, where for each query Q, predicting the probability of each schema item in the database being necessary to Q, and filter out superfluous tables and columns.

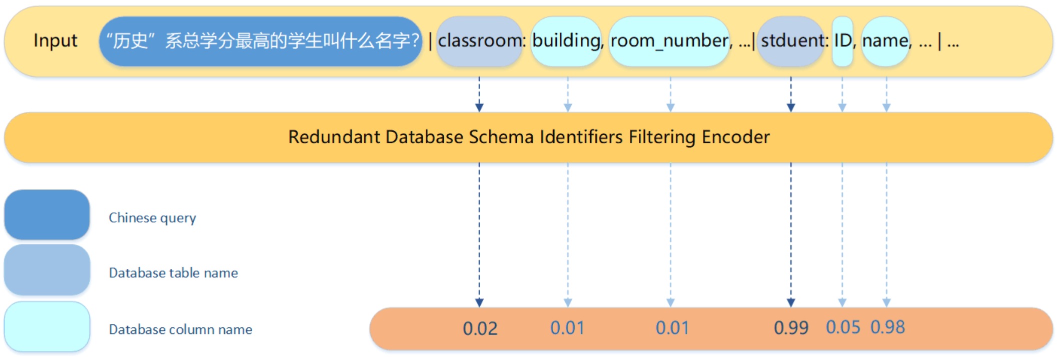

Input
“历史”系总学分最高的学生叫什么名字?
| classroom: building, room_number, ...| stduent: ID, name, ... | ...
Redundant Database Schema Identifiers Filtering Encoder
Chinese query
Database table name
Database column name
0.02
0.01
0.01
0.99
0.05
0.98

Figure 2. Schematic of the Redundant Database Schema Items Filtering Encoder.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

3.1.1. Organization Method of Input Data

In order to make the classifier fully capture the correspondence between question Q and the schema item, the Chinese semantics of the schema item and the data type of the column are taken as part of the input. Specifically, We organize the input to the model in the following form: $Input = \{Q|t_1, s_{t_1} : c_1, s_{c_1}, l_{c_1}; c_2, s_{c_2}, l_{c_2}; ...; c_{n_1}, s_{c_{n_1}}, l_{c_{n_1}}.|t_2, s_{t_2} : ...\}$, Where n_i represents the number of columns contained in table t_i, s represents the Chinese semantic information corresponding to the schema item, and l represents the data type corresponding to the column, such as NUMBER and TEXT.

we input the combination of query and database schema items into XLM-RoBERTa, a cross-lingual pre-trained language model. The RDSiF-Encoder structure is detailed in Figure 3.

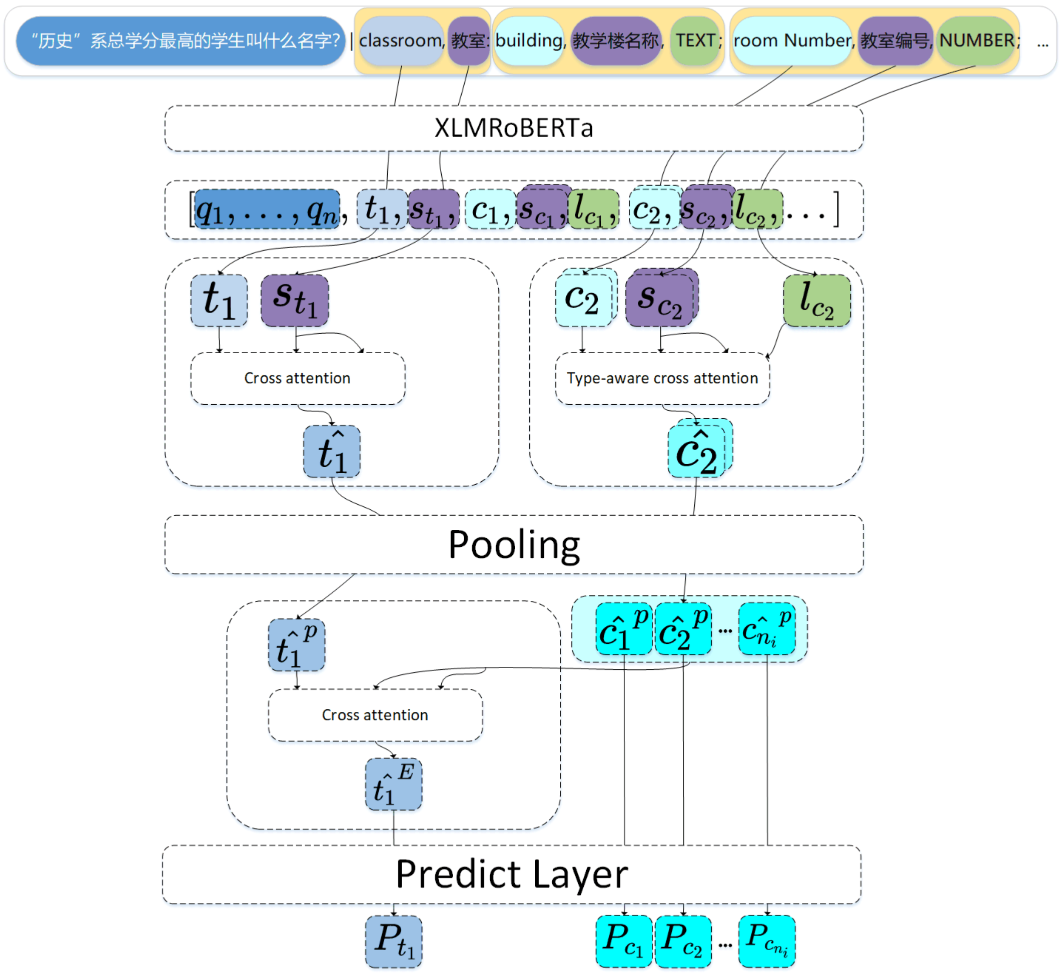

“历史”系总学分最高的学生叫什么名字?
classroom, 教室; building, 教学楼名称, TEXT; room Number, 教室编号, NUMBER; ...
XLMRoBERTa
$[q_1, \ldots, q_n, t_1, s_{t_1}, c_1, s_{c_1}, l_{c_1}, c_2, s_{c_2}, l_{c_2}, \ldots]$
t_1
s_{t_1}
Cross attention
$\hat{t_1}$
c_2
s_{c_2}
l_{c_2}
Type-aware cross attention
$\hat{c_2}$
Pooling
$\hat{t_1}^p$
$\hat{c_1}^p$
$\hat{c_2}^p$
$\hat{c_{n_i}}^p$
Cross attention
$\hat{t_1}^E$
Predict Layer
P_{t_1}
P_{c_1}
P_{c_2}
$P_{c_{n_i}}$

Figure 3. The specific structure diagram of RDSiF-Encoder. There are stacked color blocks such as c_2 and s_{c_2} to indicate that the corresponding entry has multiple tokens after tokenization. For example, the column "room number" is divided into "room" and "number".

3.1.2. Chinese Information Injected Layer for Schema Items

For the Chinese Text-to-SQL task, there is a noteworthy aspect: schema items are predominantly named in English, often with the inclusion of abbreviations, while the queries are presented in Chinese. The cross-linguistic nature and the abstract expression of schema items undoubtedly increase the difficulty of classification.

The omission of schema items in the Text-to-SQL mapping inevitably hinders generative models from capturing potential correct schema items. Thus, it is necessary to enhance cross-language schema linking to address this limitation.

Therefore, we designed Chinese information injected layer(CII layer for short) to Inject the Chinese semantic information of schema items into their corresponding original embedding. Even if the original

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

names of the database schema items are abstract, the Chinese information injection layer can help the filtering encoder to capture the schema items involved in the natural language query.

$$\begin{aligned} e_i^{(h)} &= \frac{t_i W_Q^{(h)} (s_{t_i} W_K^{(h)})}{\sqrt{d_{t_i'}/H}}; & \alpha_i^{(h)} &= softmax(e_i^{(h)}) \\ t_i'^{(h)} &= \alpha_i^{(h)} (s_{t_i} W_V^{(h)}); & t_i' &= concat(t_i'^{(1)}, t_i'^{(2)}, ..., t_i'^{(H)}) \\ \hat{t}_i &= LayerNorm(t_i + t_i'). \end{aligned} \tag{1}$$

Here, the multi-head scaled dot-product attention mechanism [30] combines the schema item embedding t_i with its Chinese semantic information s_{t_i} to capture the associations between the database schema and the Chinese natural language description. This mechanism computes weighted similarities between t_i and s_{t_i}, using these similarities as attention scores to guide the blending of information, and finally get the Chinese sematic injected schema embedding t'.

The multi-head mechanism (with H heads, and $h \in \{1, 2, ..., H\}$) enables the model to learn information in parallel across different subspaces, with each head capturing different dimensions of semantic relevance. This enriches the semantic features captured by the model.

Subsequently, Chinese sematic injected schema embedding t' is combined with the original embedding t_i through residual connection, followed by layer normalization, to ensure training stability and to retain the original information. The resulting $\hat{t}_i$ is an enhanced embedding representation infused with Chinese semantic information, which improves the model's ability to recognize and understand abstract database schema items.

3.1.3. Type-Aware Cross Attention for Columns

Query often involve the values of a certain column. According to the data type of the value, columns with mismatched data types will be filtered out. In order to enhance the ability of the classifier to perceive the data Type of the column, Type-aware cross attention is designed based on the CII layer. Specifically, the implementation process is as follows.

$$\begin{aligned} e_i^{(h)} &= \frac{c_i W_Q^{(h)} ((s_{c_i} + l_{c_i}) W_K^{(h)})}{\sqrt{d_{c_i'}/H}}; & \alpha_i^{(h)} &= softmax(e_i^{(h)}) \\ c_i'^{(h)} &= \alpha_i^{(h)} ((s_{c_i} + l_{c_i}) W_V^{(h)}); & c_i' &= concat(c_i'^{(1)}, c_i'^{(2)}, ..., c_i'^{(H)}) \\ \hat{c}_i &= LayerNorm(c_i + c_i'). \end{aligned} \tag{2}$$

The realization process of type-aware cross attention is similar to that of CII layer. The difference is that the Chinese semantic information and data type information are added, and the representation capability of the output column embedding $\hat{c}_i$ is enhanced through the integration, so that the classifier can perceive the data Type of the input column.

3.1.4. Column Injected Layer for Tables

The information involved in Q may not involve table names, which prevents the classifier from correctly classifying potential tables. For example, column c, which is necessary for Q, belongs to Table t, but Table t is not explicitly mentioned in Q. Drawing on the work of [3], we designed a column injected layer to enhance the relevance of tables to their columns.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

$$
\begin{aligned}
&C_i = concat(\hat{c_1}^p, \hat{c_2}^p, ..., \hat{c_{n_i}}^p); \\
&e_i^{(h)} = \frac{\hat{f_i}^P W_Q^{(h)} (C_i W_K^{(h)})}{\sqrt{d_{\hat{f_i}^{p'}} / H}}; \qquad \alpha_i^{(h)} = softmax(e_i^{(h)}) \\
&\hat{f_i}^{p'(h)} = \alpha_i^{(h)} (C_i W_V^{(h)}); \qquad \hat{f_i}^{p'} = concat(\hat{f_i}^{p'(1)}, \hat{f_i}^{p'(2)}, ..., \hat{f_i}^{p'(H)}) \\
&\hat{f_i}^E = LayerNorm(\hat{f_i}^p + \hat{f_i}^{p'}).
\end{aligned} \tag{3}
$$

$\hat{f_i}^E$ is the enhanced table embedding that injects information about the columns it contains.

3.1.5. Tables and Columns Classifier

To complete the binary classification task, two fully connected layers are set up at the end of the encoder.

$$P_x = FC_2(LeakeyReLU(FC_1(x))) \tag{4}$$

Where x refers to the enhanced table embedding $\hat{f_i}^E$ or the pooled type-aware column embedding $\hat{c_i}^p$.

3.2. Generative Pre-Train LLM for SQL Parser

Language models that have undergone pre-training have learned a wealth of linguistic knowledge from large corpora, better capturing the diversity and fluctuation of natural language and understanding the contextual relationships within. This ability to comprehend context allows the generative SQL parser to adapt more flexibly to various language expressions and query structures, better handling complex natural language queries.

Using open-source LLMs to tackle Text-to-SQL tasks provides more room for optimization, achieving an advantage in parameter scale while ensuring performance.

Before conducting supervised fine-tuning for the Language Model (LLM), it is essential to prepare training data. In the context of Text-to-SQL, particular attention should be given to the method of representing questions. In question representation, a primary challenge is the effective combination of natural language questions with database schema information. The goal of question representation is to control the probability of the LLM generating SQL statements, maximizing the likelihood of generating correct SQL queries.

$$\max_f P_{\mathcal{G}}(S^* | f(Q, D))$$

where $P_{\mathcal{G}}(S^* | f(q, D))$ denotes the probability of generating the correct SQL query S^* given the query Q and the associated database schema information D. The function f encapsulates the question representation process, and the optimization aims to maximize this conditional probability, ensuring the generation of accurate SQL queries by the LLM.

To organize the training and test data required for LLM supervised fine-tuning, we proposed a question representation method: IECQN.

In IECQN, first, the instruction(I) articulates the task to be solved by the SQL parser and the output standard; Then, an example(E) output helps the SQL parser clarify the output standard further; The chosen database schema information(C) is expressed through listing the N most relevant tables and M columns within those tables associated with the question; Next, the natural language query(Q) is presented. Finally, NatSQL(N) is used as an intermediary representation to bridge the gap between

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

natural language and SQL queries. This means using the NatSQL corresponding to the original gold SQL query as the target response.

Most existing methods for LLMs focusing on prompt engineering and in-context learning do not explore the potential of supervised fine-tuning; The goal of supervised fine-tuning is to minimize the following empirical loss:

$$\min_{\mathcal{G}^*} \sum_{i=1}^{|\mathcal{T}|} \mathcal{L}(\mathcal{G}^*(f(q_i, D_i)), S_i)$$

Here, the goal is to find the best-performing parameters $\mathcal{G}^*$ which minimize the accumulated loss over the training set $\mathcal{T} = (q_i, D_i, S_i)$. The function f takes query q_i and its corresponding database information D_i, encapsulates the question representation process. $\mathcal{G}^*(f(q_i, D_i))$ denotes the SQL query generated by $\mathcal{G}^*$. The loss function $\mathcal{L}$ evaluates the deviation of this prediction from the gold annotation S_i. Through this optimization, the fine-tuned model is better positioned to capture the nuances of the targeted tasks and perform with greater accuracy.

As pretrained language models grow larger, the difficulty of traditional fine-tuning increases. Considering computational resources and training costs, we opted for parameter-efficient fine-tuning. The trainable parameters in some parameter-efficient fine-tuning (PEFT) methods are significantly fewer compared to the parameter scale of a LLM [31–33].

The fine-tuned $\mathcal{G}^*$ is used for inference, creating NatSQL queries by processing the input natural language question and the specified database. The question representation method used during inference is consistent with the method employed to organize the training data.

SQL Parser first generates intermediate representation NatSQL, which can be translated to SQL queries via a non-trainable transpiler [25]. During inference, a beam search of size B is performed, with the first executable SQL query output to ensure execution accuracy.

We conducted a series of experiments to validate the effectiveness of our proposed question representation method IECQN, while also discussing the vast potential of supervised fine-tuning on LLMs for Text-to-SQL tasks.

3.3. SQL Query Correcting Decoder

A widely recognized downside of multi-stage pipeline methods is error propagation, where incorrect categorization of redundant items inevitably affects the generative capabilities of large language models for SQL query generation. Hence, we collected correct and incorrect inference results from G_i (i denotes a specific parameter-efficient fine-tuning) to train an SQL query correcting decoder, mitigating the effects of error propagation. Given a natural language query Q, database schema information D, and the erroneous SQL query S^-, the goal of the correcting decoder is to predict the edit operations m required for correction and the corrected SQL S^+. Following existing works [34], we defined the correction task as a sequence-to-sequence generation:

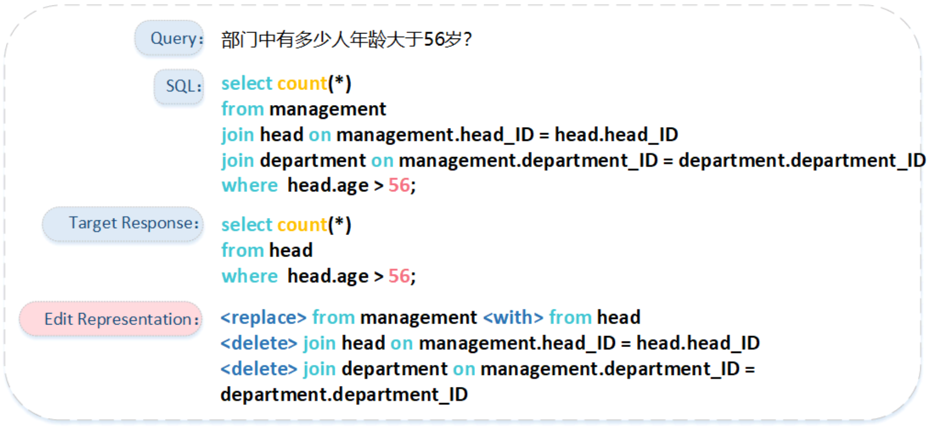
$$P((m, S^+)|(Q, D, S^-)) = \prod_{t=1}^{T} P((m_t, S_t)|((Q, D, S^-), (m_{1:t-1}, S_{1:t-1}))) \tag{5}$$

The equation represents the probability of correctly predicting each edit operation m_t and the updated SQL snippet S_t at time step t, conditioned on the input query Q, schema D, the erroneous SQL S^-, and the sequence of all previously predicted edits and SQL snippets up to time $t-1$. This sequence-to-sequence generation framework serves to iteratively refine the erroneous query towards the correct form, ultimately producing the corrected SQL S^+.

To ensure that the training data for the correcting decoder reflects the true distribution of errors, we performed k-fold cross-validation on the Chinese Text-to-SQL dataset using a specific pretrained LLM, synthesizing a correction dataset.

Specifically, for the training set, the Chinese Text-to-SQL training set was divided into k parts, using k-1 subsets to supervise fine-tune the LLM while evaluating on the remaining subset. If the parsing result is not an exact set match or execution match to the gold annotation, it is marked as erroneous; otherwise, it is deemed correct and included in the correction training set. For the test set, LLM is supervised fine-tuned on the training set of the complete Chinese Text-to-SQL dataset to obtain $\mathcal{G}^*$, and then $\mathcal{G}^*$ is evaluated on the development set of the Chinese Text-to-SQL dataset. Then the evaluation results are included in the test set of the error correction data set. The training data of organization and fine-tuning methods used for the supervised fine-tuning of LLM are consistent with those mentioned in Section 3.2. The specific pre-trained LLM refers to the pre-trained LLM used by the Generative Pre-train LLM SQL Parser.

We used three editing operations: replace, insert, and delete, to correct the erroneous SQL queries in the correction dataset relative to the target output, completing the construction of the correction dataset.



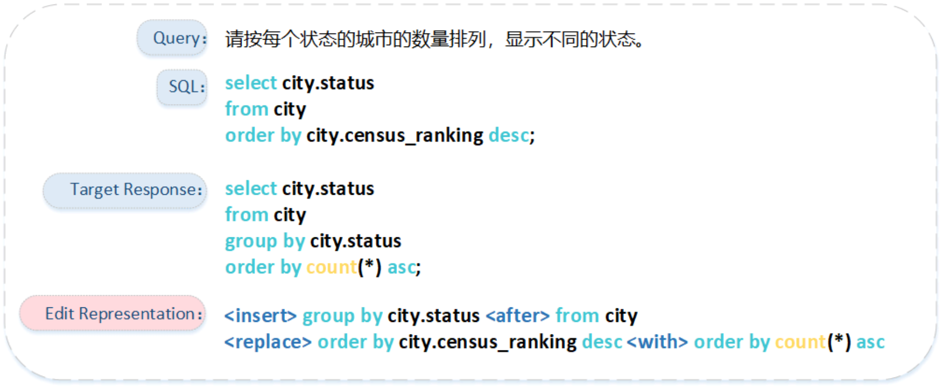
Query: 部门中有多少人年龄大于56岁?

SQL:
```
select count(*)
from management
join head on management.head_ID = head.head_ID
join department on management.department_ID = department.department_ID
where  head.age > 56;
```

Target Response:
```
select count(*)
from head
where  head.age > 56;
```

Edit Representation:
```
<replace> from management <with> from head
<delete> join head on management.head_ID = head.head_ID
<delete> join department on management.department_ID =
department.department_ID
```

(a)



Query: 请按每个状态的城市的数量排列，显示不同的状态。

SQL:
```
select city.status
from city
order by city.census_ranking desc;
```

Target Response:
```
select city.status
from city
group by city.status
order by count(*) asc;
```

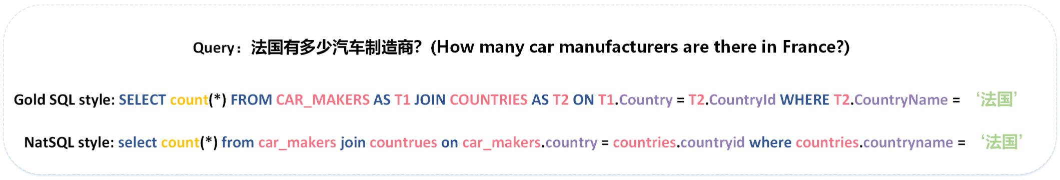
Edit Representation:
```
<insert> group by city.status <after> from city
<replace> order by city.census_ranking desc <with> order by count(*) asc
```

(b)

Figure 4. a and b show two examples collected through cross-validation using LlaMa2-7b-chat on the CSpider training set. These examples illustrate three types of editing operations: replace, insert, and delete. These operations are performed to correct errors in SQL queries relative to the target response in the correction dataset.

It is worth noting that the CSpider is translated from the Spider dataset, which was annotated by eleven different annotators to extract gold SQL queries, with inconsistent annotation styles.

Also, the IECQN representation method we proposed in Section 3.2 uses NatSQL as an intermediate representation. When the intermediate representation is converted to SQL queries, its style (NatSQL style for short) has certain differences from the gold SQL queries. Differences in style see the diagram below.



Query：法国有多少汽车制造商? (How many car manufacturers are there in France?)

Gold SQL style: `SELECT count(*) FROM CAR_MAKERS AS T1 JOIN COUNTRIES AS T2 ON T1.Country = T2.CountryId WHERE T2.CountryName = '法国'`

NatSQL style: `select count(*) from car_makers join countrues on car_makers.country = countries.countryid where countries.countryname = '法国'`

Figure 5. Illustration of differences between two kind of SQL in style.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

Gold SQL Style tends to use uppercase letters to represent keywords, table names, and column names, and employs the AS keyword to alias tables. In contrast, NatSQL Style leans towards the use of lowercase letters, directly utilizing table names without the AS keyword.

The construction rules for these two style SQL queries are inconsistent, and the ultimate output of the generative NatSQL parser is the former. Considering all the above factors, to reduce the learning difficulty of the correction model and avoid conflicts in the construction rules between the erroneous SQL queries and the target outputs, when correcting the erroneous SQL queries, the NatSQL style SQL query corresponding to the original gold SQL in the CSpider dataset is taken as the target output. In this way, we can reduce the risk of erroneous corrections due to style differences, enhancing the robustness and accuracy of the correction decoder when dealing with actual SQL queries.

4. Experimental Setup

4.1. Data and Environment

CSpider

We conducted a series of experiments using the CSpider dataset. CSpider is a large-scale Chinese dataset for complex cross-domain semantic parsing and Text-to-SQL tasks, comprising the entire dataset translated from Spider for Chinese Text-to-SQL tasks. The CSpider dataset consists of a training set with 7000 samples and a development set with 1034 samples, with no overlap between the databases involved in the training and development sets.

IECQN representation dataset for the SFT of LLM

The IECQN representation consists of a concatenation of instructions, examples, chosen schema items, natural language questions, and NatSQL intermediate representations. The dataset organization method for the IECQN representation has already been introduced in Section 3.2. The natural language questions are sourced from questions in the CSpider dataset, and the chosen schema items come from the output of the trained redundant item filtering encoder. The two parameters N and M for the chosen schema items are set to 4 and 5. It is observed that the SQL query statements in the CSpider dataset mostly involve fewer than four tables (with only 32 samples involving five tables in the training set), we set the number of most relevant tables in the IECQN representation to 4, and similarly, the number of the most relevant columns in the chosen tables to 5, to avoid interference from superfluous items on LLM performance, while ensuring that the chosen database schema items cover the items mentioned in the natural language questions as much as possible.

SQL Query Correction Dataset The preparation steps for the correction dataset are specifically elaborated in Section 3.3, which is not reiterated here. The Chinese Text-to-SQL dataset mentioned refers to the CSpider dataset.

All experiments were carried out on a server with two NVIDIA 3090 (24G) GPUs, an Intel Xeon Gold 6230 CPU, 125GB of memory, and an Ubuntu 20.04 LTS operating system.

4.2. Evaluation Metrics

To evaluate the performance of the redundant item filtering encoder, we use the Area Under the ROC Curve (AUC) as the evaluation metric. AUC is calculated separately for table classification and column classification.

To assess the performance of our proposed method, we adopt two metrics: Exact Match Accuracy and Exact Execution Accuracy. Exact Match Accuracy measures whether an SQL query can accurately match the gold SQL query by converting the predicted SQL query into a special data structure. Exact Execution Accuracy compares the execution result of the predicted SQL query with that of the gold SQL query.

4.3. Details

The training of FGCSQL is divided into three phases.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

In the first phase, the redundant item filtering encoder is trained with the number of heads h for the Chinese information enhancement layer set to 8. It is optimized using an AdamW optimizer with a batch size of 8 and a learning rate of 1e-5.

In the second phase, for training the generative SQL decoder, we conducted comparative experiments on pre-trained LLMs like LlaMa2-7b-chat [35], ChatGLM2-6b [36], ChatGLM3-6b [36], BaiChuan2-7b [37], and eventually selected LlaMa2-7b-chat as the best model variant for the SQL parser after supervised fine-tuning due to its optimal comprehensive performance. The parameter-efficient fine-tuning method used was LoRA [38], with lora_rank set to 64 and lora_alpha set to 32, utilizing cosine annealing with restarts to adjust the learning rate, and the beam size B set to 16 for beam search.

In the third phase, training the correcting decoder. We selected LlaMa2-7b-chat for the SQL parser, and CodeT5 [39] for the error correction models. The batch size is set to 4, with a learning rate for Adafactor of 3e-5, using linear decay to adjust the learning rate.

5. Result

5.1. Results on CSpider

We compared our proposed method with the following methods: RAT-SQL [4], LGESQL [40], FastRAT [41], ChatGPT [9], Heterogeneous Graph + Relative Position Attention [42], RESDSQL [3], Auto-SQL-Correction [34].

On the development set of the CSpider dataset, the performance of FGCSQL and other methods on the EM and EX evaluation metrics is shown in Table 1.

Table 1. EM and EX results on CSpider's development set. We compare our approach with some powerful baseline methods. Where, BS stands for basic problem representation [12].

Approach	EM	EX
RAT-SQL	41.4	-
LGESQL	58.6	-
FastRAT	61.3	67.7
ChatGPT + BS	32.6	65.1
Heterogeneous Graph + Relative Position Attention	66.2	-
RESDSQL + NatSQL	65.6	79.1
Auto-SQL-Correction + NatSQL	64.6	75.9
Ours	65.8	81.1

Our best model variant, FGCSQL, performs closely to the state-of-the-art (SOTA) on the EM metric (-0.4% difference), and it outperforms all baseline variants on the EX metric. This indicates that our three-stage pipeline method can significantly reduce the learning difficulty of Text-to-SQL conversion. Our FGCSQL achieved competitive performance, improving EX from 79.1% to 81.1%, showcasing the effectiveness of our method. Notably, all model variants that combine with NatSQL show significant improvement on the EX metric compared to previous studies, as using NatSQL as an interim representation bridges the gap between natural language and SQL queries, further reducing learning difficulty.

5.2. Ablation Studies

To analyze the effectiveness of each design component, we conducted a series of ablation experiments on the CSpider development set.

5.2.1. Effect of Chinese Information Injection Layer

The design of the Chinese Information Injection Layer aims to address the difficulties of cross-language schema linking and reduce the parsing difficulty for the subsequent generative SQL parser. AUC is calculated separately for table and column classification with and without the use of the

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

Chinese Information Injected Layer and Type-aware attention, with experimental results shown in Table 2.

Table 2. Ablation studies of Chinese Information Injected Layer.

Model Variant	Table AUC	Column AUC	Total
RDsiF-Encoder-without CII layer	0.9725	0.9703	1.9428
RDsiF-Encoder-without Type-aware attention	0.9811	0.9713	1.9524
RDsiF-Encoder	0.9814	0.9795	1.9609

The CII layer improves the filtering encoder's table classification AUC and column classification AUC by 0.89% and 0.92%, respectively. Type-awared attention improved the filter encoder's column classification AUC by 0.82%. The enhanced performance of AUC indicates that CII layer is helpful to capture the mapping features between Chinese information and corresponding English schema items, and Type-awared attention improves the classifier's Type perception ability of column information.

5.2.2. Effect of Redundant Database Schema Items Filtering Encoder

The design of the Redundant Database Schema Items Filtering Encoder aims to resolve the issues with end-to-end methods increasing the learning difficulty of Text-to-SQL. By filtering the database schema items corresponding to the natural language Queries in CSpider through the Redundant Item Filtering Encoder, the most relevant database schema items are provided for the IECQN query representation. Table 3 shows the performance comparison between using filtered database schema items and using all schema items as database information in the question representation, for LlaMa2-7b-chat and ChatGLM2-6b following Lora fine-tuning.

Table 3. Ablation studies of Redundant Database Schema Items Filtering Encoder.

Model Variant	EM	EX
LlaMa2-7b-chat + LoRA + IECQN	65.1	80.9
LlaMa2-7b-chat + LoRA + IECQN -with all schema items	61.1	73.0
ChatGLM2-6b + LoRA + IECQN	64.2	79.7
ChatGLM2-6b + LoRA + IECQN -with all schema items	59.3	71.4

For the fine-tuned LlaMa2-7b-chat as the SQL parser, using the filtered database schema item information led to a 4.0% increase in EM and a 7.9% increase in EX.

For the fine-tuned ChatGLM2-6b as the SQL parser, using the filtered database schema item information led to a 4.9% increase in EM and an 8.3% increase in EX.

By filtering out irrelevant information that would negatively impact the SQL parser using the Redundant Item Filtering Encoder, the fine-tuned LLM experienced a significant performance boost.

5.2.3. Effect of IECQN Question Representation

By using IECQN question representation, an effective expression of natural language questions and database schema information increases the likelihood of generating the correct SQL queries. To validate the effectiveness of IECQN query representation, we compared the parser trained with Basic Question Representation (BS) to one trained with IECQN representation, and experimental results shown in Table 4.

Table 4. Ablation studies of IECQN Question Representation.

Model Variant	EM	EX
LlaMa2-7b-chat + LoRA + IECQN	65.1	80.9
LlaMa2-7b-chat + LoRA + BS	53.1	68.0
ChatGLM2-6b + LoRA + IECQN	64.2	79.7
ChatGLM2-6b + LoRA + BS	36.6	51.1

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

Compared to Basic Question Representation training, using IECQN representation led to a 12.0% increase in EM and 12.9% increase in EX for the fine-tuned LlaMa2-7b-chat as the SQL parser. For the fine-tuned ChatGLM2-6b as the SQL parser, a 27.6% increase in EM and 28.6% increase in EX were observed.

This is because the database information in IECQN representation is customized for the natural language question, eliminating interference from redundant information. On the other hand, using NatSQL as the target output narrows the gap between natural language queries and SQL statements, further lowering the learning difficulty of Text-to-SQL.

5.2.4. Effect of Parameter-Efficiently Fine-Tuned LLM

The purpose of using a parameter-efficiently fine-tuned LLM as the SQL parser is to explore the potential of open-source LLM in Chinese Text-to-SQL tasks, achieving comparable or even superior performance to larger scale language models with lower training costs.

We compared the performance of LoRA fine-tuned LlaMa2-7b-chat and ChatGLM2-6b with Basic Question Representation with the original un-fined-tuned models. Evaluations used the same input context organization method: BS. Given ChatGPT's excellent performance on various natural language tasks, we also added comparison results with ChatGPT. Table 5 shows experimental results.

Table 5. Ablation studies of Parameter-Efficiently Fine-tuned LLM.

Model Variant	EM	EX
ChatGPT	32.6	65.1
LlaMa2-7b-chat	14.3	23.4
LlaMa2-7b-chat + LoRA + BS	53.1	68.0
ChatGLM2-6b	13.1	21.9
ChatGLM2-6b + LoRA + BS	36.6	51.1

Parameter-efficiently fine-tuned LlaMa2-7b-chat improved by 38.8% in EM and 44.6% in EX. Parameter-efficiently fine-tuned ChatGLM2-6b improved by 38.8% in EM and 44.6% in EX as well. Compared to the closed-source ChatGPT, parameter-efficiently fine-tuned LlaMa2-7b-chat was 20.5% ahead in EM and 2.9% ahead in EX. The significant gap in EM compared to EX is because EM is insensitive to the execution results of generated SQL; sometimes the correct SQL for the same Query varies substantially in expression order and logic, with EM occasionally misclassifying correct SQL as wrong, leading to false negatives. This means some SQL queries generated by ChatGPT are more diverse and flexibly expressed, yet the execution results are accurate.

The parameter-efficiently fine-tuned open-source LLM achieved comparable performance in Chinese Text-to-SQL tasks, particularly the parameter-efficiently fine-tuned LlaMa2-7b-chat, which surpassed the 20b-parameter ChatGPT in both metrics.

5.2.5. Effect of SQL Query Correcting Decoder

SQL Query Correcting Decoder aims to resolve the error propagation problem inherent in multi-stage pipeline methods. In our experiments, the LLM used in the generative SQL parser was LlaMa2-7b-chat. We compared the output SQL queries from the first two stages of FGCSQL with the complete FGCSQL, with results presented in Table 6

Table 6. Ablation studies of SQL Query Correcting Decoder.

Model Variant	EM	EX
FGCSQL	65.8	81.1
FGCSQL -without SQL Query Correcting Decoder	65.1	80.9

The use of the Correction Decoder led to a 0.7% improvement in EM, and a not significant 0.2% improvement in EX, nonetheless, achieving the best performance on the CSpider dataset. This is

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

because the correction decoder somewhat shields against the negative impact that misclassification of redundant items could have on generative LLM SQL query generation, reducing error propagation, and on the other hand, reduces the risks of incorrect corrections due to style differences, enhancing the robustness and accuracy of the correction decoder when handling practical SQL queries.

6. Conclusion

This study introduces FGCSQL, a three-stage pipeline method for open-source large language model (LLM)-driven Chinese Text-to-SQL transformation. The proposed pipeline includes a redundant database schema items filtering encoder, a generative pre-trained LLM for SQL parsing, and an SQL query correcting decoder. These components are designed to address the challenges associated with Text-to-SQL tasks, particularly those involving cross-lingual schema linking, the complexity of queries, and error propagation within multi-stage processing.

The FGCSQL model demonstrates competitive performance on the CSpider dataset, a large-scale Chinese benchmark for cross-domain semantic parsing and Text-to-SQL tasks. The model achieves near state-of-the-art (SOTA) performance on Exact Match (EM) and surpasses all baseline variants on Exact Execution (EX), showing that the three-stage pipeline method significantly reduces the learning difficulty of Text-to-SQL translation.

Extensive ablation studies and comparison with various baseline approaches, including methods like LlaMa2-7b-chat with Basic Question Representation, RAT-SQL, LGESQL, and others, highlight the effectiveness of each component within FGCSQL. Notable findings include the impact of Chinese information injected layer and Type-aware attention on improving schema item classification, and the effectiveness of the IECQN query representation, which customizes database schema information to natural language queries and uses NatSQL as an intermediate representation, further minimizing learning difficulty.

FGCSQL boasts superior execution precision compared to similar methods like ChatGPT, and displays the advantage of leveraging open-source LLMs, particularly those with parameter-efficient fine-tuning, which significantly outperform larger closed-source models like ChatGPT in both EM and EX.

Lastly, the SQL query correcting decoder, designed to mitigate error propagation, showcases incrementally improved EM and benchmarks as the best performance on the CSpider dataset. The correction decoder demonstrates increased robustness and accuracy in dealing with practical SQL queries.

Author Contributions: Conceptualization, G.J.; Methodology, G.J. and W.L.; Validation, G.J. and C.Y.; Writing—original draft, G.J.; Writing—review & editing, G.J and Z.Z.; Visualization, G.J. and W.L. All authors have read and agreed to the published version of the manuscript.

Funding: The support projects for this work include Xianyang City Key Research and Development Plan Project (2021ZDYF-GY-0031), Shaanxi Provincial Department and City Joint Key Project (2022GD-TSLD-61-3), Xi'an City Science and Technology Plan Project (23ZDCYTSGG0026-2022), Shanxi Coal Geology Group Co., Ltd. Scientific Research Project (SMDZ-2023CX-14), and Shaanxi Provincial Water Conservancy Development Fund Science and Technology Project (2024SLKJ-16).

Data Availability Statement: The CSpider dataset is available at https://github.com/taolusi/chisp (accessed on 8 December 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

References

1. Zheng, Y.; Wang, H.; Dong, B.; Wang, X.; Li, C. HIE-SQL: History Information Enhanced Network for Context-Dependent Text-to-SQL Semantic Parsing. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2022; Muresan, S.; Nakov, P.; Villavicencio, A., Eds., Dublin, Ireland, 2022; pp. 2997–3007.
2. Hui, B.; Geng, R.; Wang, L.; Qin, B.; Li, Y.; Li, B.; Sun, J.; Li, Y. S^{2}SQL: Injecting Syntax to Question-Schema Interaction Graph Encoder for Text-to-SQL Parsers. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2022; Muresan, S.; Nakov, P.; Villavicencio, A., Eds., Dublin, Ireland, 2022; pp. 1254–1262. https://doi.org/10.18653/v1/2022.findings-acl.99.
3. Li, H.; Zhang, J.; Li, C.; Chen, H. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence, 2023, Vol. 37, pp. 13067–13075.
4. Wang, B.; Shin, R.; Liu, X.; Polozov, O.; Richardson, M. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In Proceedings of the Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics; Jurafsky, D.; Chai, J.; Schluter, N.; Tetreault, J., Eds., Online, 2020; pp. 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677.
5. Price, P.J. Evaluation of Spoken Language Systems: the ATIS Domain. In Proceedings of the Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990, 1990.
6. Hazoom, M.; Malik, V.; Bogin, B. Text-to-SQL in the Wild: A Naturally-Occurring Dataset Based on Stack Exchange Data. In Proceedings of the Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021); Lachmy, R.; Yao, Z.; Durrett, G.; Gligoric, M.; Li, J.J.; Mooney, R.; Neubig, G.; Su, Y.; Sun, H.; Tsarfaty, R., Eds., Online, 2021; pp. 77–87. https://doi.org/10.18653/v1/2021.nlp4prog-1.9.
7. Zhong, V.; Xiong, C.; Socher, R. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning, 2017, [arXiv:cs.CL/1709.00103].
8. Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; et al. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In Proceedings of the Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing; Riloff, E.; Chiang, D.; Hockenmaier, J.; Tsujii, J., Eds., Brussels, Belgium, - 2018; pp. 3911–3921. https://doi.org/10.18653/v1/D18-1425.
9. Liu, A.; Hu, X.; Wen, L.; Yu, P.S. A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability. *arXiv preprint arXiv:2303.13547* **2023**.
10. Rajkumar, N.; Li, R.; Bahdanau, D. Evaluating the Text-to-SQL Capabilities of Large Language Models, 2022, [arXiv:cs.CL/2204.00498].
11. Trummer, I. CodexDB: Generating Code for Processing SQL Queries using GPT-3 Codex, 2022, [arXiv:cs.DB/2204.08941].
12. Pourreza, M.; Rafiei, D. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction, 2023, [arXiv:cs.CL/2304.11015].
13. Gao, D.; Wang, H.; Li, Y.; Sun, X.; Qian, Y.; Ding, B.; Zhou, J. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation, 2023, [arXiv:cs.DB/2308.15363].
14. Yu, T.; Li, Z.; Zhang, Z.; Zhang, R.; Radev, D. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation. In Proceedings of the Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers); Walker, M.; Ji, H.; Stent, A., Eds., New Orleans, Louisiana, 2018; pp. 588–594. https://doi.org/10.18653/v1/N18-2093.
15. Liu, Q.; Yang, D.; Zhang, J.; Guo, J.; Zhou, B.; Lou, J.G. Awakening Latent Grounding from Pretrained Language Models for Semantic Parsing. In Proceedings of the Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021; Zong, C.; Xia, F.; Li, W.; Navigli, R., Eds., Online, 2021; pp. 1174–1189. https://doi.org/10.18653/v1/2021.findings-acl.100.
16. Song, H.; Zhang, C.; Li, Q.; Song, D. An end-to-end multi-task learning to link framework for emotion-cause pair extraction. *2021 International Conference on Image, Video Processing, and Artificial Intelligence* **2020**.
17. Chen, Z.; Huang, H.; Liu, B.; Shi, X.; Jin, H. Semantic and Syntactic Enhanced Aspect Sentiment Triplet Extraction, 2021, [arXiv:cs.CL/2106.03315].
18. Liu, Y.; Zhang, J.; Xiong, H.; Zhou, L.; He, Z.; Wu, H.; Wang, H.; Zong, C. Synchronous Speech Recognition and Speech-to-Text Translation with Interactive Decoding. *CoRR* **2019**, *abs/1912.07240*, [1912.07240].

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

19. Deng, X.; Awadallah, A.H.; Meek, C.; Polozov, O.; Sun, H.; Richardson, M. Structure-Grounded Pretraining for Text-to-SQL. *CoRR* **2020**, *abs/2010.12773*, [2010.12773].
20. Ma, C.; Zhang, W.; Huang, M.; Feng, S.; Wu, Y. Integrating Relational Structure to Heterogeneous Graph for Chinese NL2SQL Parsers. *Electronics* **2023**, *12*. https://doi.org/10.3390/electronics12092093.
21. Bogin, B.; Berant, J.; Gardner, M. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. In Proceedings of the Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics; Korhonen, A.; Traum, D.; Màrquez, L., Eds., Florence, Italy, 2019; pp. 4560–4565. https://doi.org/10.18653/v1/P19-1448.
22. Yu, T.; Yasunaga, M.; Yang, K.; Zhang, R.; Wang, D.; Li, Z.; Radev, D. SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task. In Proceedings of the Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing; Riloff, E.; Chiang, D.; Hockenmaier, J.; Tsujii, J., Eds., Brussels, Belgium, - 2018; pp. 1653–1663. https://doi.org/10.18653/v1/D18-1193.
23. Choi, D.; Shin, M.C.; Kim, E.; Shin, D.R. RYANSQL: Recursively Applying Sketch-based Slot Fillings for Complex Text-to-SQL in Cross-Domain Databases. *Computational Linguistics* **2021**, *47*, 309–332.
24. Guo, J.; Zhan, Z.; Gao, Y.; Xiao, Y.; Lou, J.G.; Liu, T.; Zhang, D. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In Proceedings of the Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics; Korhonen, A.; Traum, D.; Màrquez, L., Eds., Florence, Italy, 2019; pp. 4524–4535. https://doi.org/10.18653/v1/P19-1444.
25. Gan, Y.; Chen, X.; Xie, J.; Purver, M.; Woodward, J.R.; Drake, J.; Zhang, Q. Natural SQL: Making SQL Easier to Infer from Natural Language Specifications. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2021; Moens, M.F.; Huang, X.; Specia, L.; Yih, S.W.t., Eds., Punta Cana, Dominican Republic, 2021; pp. 2030–2042. https://doi.org/10.18653/v1/2021.findings-emnlp.174.
26. Scholak, T.; Schucher, N.; Bahdanau, D. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In Proceedings of the Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing; Moens, M.F.; Huang, X.; Specia, L.; Yih, S.W.t., Eds., Online and Punta Cana, Dominican Republic, 2021; pp. 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779.
27. Dou, L.; Gao, Y.; Pan, M.; Wang, D.; Che, W.; Zhan, D.; Lou, J.G. UniSAr: A Unified Structure-Aware Autoregressive Language Model for Text-to-SQL, 2022, [arXiv:cs.CL/2203.07781].
28. Wang, C.; Tatwawadi, K.; Brockschmidt, M.; Huang, P.S.; Mao, Y.; Polozov, O.; Singh, R. Robust Text-to-SQL Generation with Execution-Guided Decoding, 2018, [arXiv:cs.CL/1807.03100].
29. Hwang, W.; Yim, J.; Park, S.; Seo, M. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization, 2019, [arXiv:cs.CL/1902.01069].
30. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Advances in neural information processing systems* **2017**, *30*.
31. He, H.; Cai, J.; Zhang, J.; Tao, D.; Zhuang, B. Sensitivity-Aware Visual Parameter-Efficient Fine-Tuning, 2023, [arXiv:cs.CV/2303.08566].
32. Liao, B.; Meng, Y.; Monz, C. Parameter-Efficient Fine-Tuning without Introducing New Latency, 2023, [arXiv:cs.CL/2305.16742].
33. Lu, J.; Yu, L.; Li, X.; Yang, L.; Zuo, C. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning, 2023, [arXiv:cs.SE/2308.11148].
34. Chen, Z.; Chen, S.; White, M.; Mooney, R.; Payani, A.; Srinivasa, J.; Su, Y.; Sun, H. Text-to-SQL Error Correction with Language Models of Code, 2023, [arXiv:cs.CL/2305.13073].
35. Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023, [arXiv:cs.CL/2307.09288].
36. Du, Z.; Qian, Y.; Liu, X.; Ding, M.; Qiu, J.; Yang, Z.; Tang, J. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In Proceedings of the Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2022, pp. 320–335.
37. Baichuan. Baichuan 2: Open Large-scale Language Models. *arXiv preprint arXiv:2309.10305* **2023**.
38. Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. *CoRR* **2021**, *abs/2106.09685*, [2106.09685].
39. Wang, Y.; Wang, W.; Joty, S.; Hoi, S.C. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In Proceedings of the Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing; Moens, M.F.; Huang, X.; Specia, L.; Yih, S.W.t., Eds., Online and Punta Cana, Dominican Republic, 2021; pp. 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 February 2025 doi:10.20944/preprints202502.1059.v1

40. Cao, R.; Chen, L.; Chen, Z.; Zhao, Y.; Zhu, S.; Yu, K. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. In Proceedings of the Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers); Zong, C.; Xia, F.; Li, W.; Navigli, R., Eds., Online, 2021; pp. 2541–2555. https://doi.org/10.18653/v1/2021.acl-long.198.
41. Vougiouklis, P.; Papasarantopoulos, N.; Zheng, D.; Tuckey, D.; Diao, C.; Shen, Z.; Pan, J.Z. FastRAT: Fast and Efficient Cross-lingual Text-to-SQL Semantic Parsing. *Proc. of IJCNLP-AACL* **2023**, *2023*.
42. Ma, C.; Zhang, W.; Huang, M.; Feng, S.; Wu, Y. Integrating Relational Structure to Heterogeneous Graph for Chinese NL2SQL Parsers. *Electronics* **2023**, *12*, 2093.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.