Enhancing Text-to-SQL with Open-source LLMs

Ruilin Hu
2024210886
Tsinghua University
hrl24@mails.tsinghua.edu.cn

Lu Fan
2024210883
Tsinghua University
fanl24@mails.tsinghua.edu.cn

Yizhe Chen
2023311175
Tsinghua University
chenyizh23@mails.tsinghua.edu.cn

Abstract

The text-to-SQL task seeks to bridge natural language questions and database query systems by converting user queries into executable SQL statements. While recent advancements in large language models (LLMs) have significantly improved the task's accuracy, current methods often rely on proprietary LLMs with high costs, limited accessibility, and data privacy concerns. This paper presents SageSQL, a novel multi-agent framework leveraging open-source LLMs to address these challenges. SageSQL introduces a robust schema linking process, enabling accurate identification of relevant database components, followed by a diverse SQL generation module to maximize structural variety in generated queries. A self-consistency-based post-processing mechanism further refines the final SQL output. Experimental results on the Spider and BIRD benchmarks demonstrate that SageSQL outperforms state-of-the-art methods based on open-source LLMs and achieves competitive performance with proprietary LLMs, highlighting its potential as a cost-effective, privacy-preserving solution for complex text-to-SQL tasks.

1 Introduction

The text-to-SQL task (also known as natural language to SQL or NL2SQL) aims at translating a user's question into a valid and semantically aligned SQL query that can be executed in a given database. By enabling users without SQL expertise to interact with databases, text-to-SQL allows users to efficiently retrieve the required data from databases, perform data analysis, and finally make data-driven decisions[25, 14].

Due to the impressive capabilities of large language models (LLMs) in language understanding and code generation, numerous studies have investigated the application of LLMs to the text-to-SQL task. Although high accuracy was obtained, most existing methods based on in-context learning depend heavily on the capability of powerful proprietary LLMs, which have unignorable limitations on practical applications. (**L1**) Firstly, previous methods based on few-shots prompting[18, 7] require inputting a large number of tokens into proprietary LLMs like GPT-4, which can incur significant economic costs (for example, DIN-SQL+GPT-4[18] costs around 0.7$ per question). (**L2**) Another issue is data privacy: many companies cannot send internal database schema information or user data to LLM service providers. (**L3**) Additionally, the reliance on proprietary models accessed solely through API calls, with no visibility into their underlying architectures or inference mechanisms, significantly restricts their flexibility and efficiency in deployment for various application environments. These limitations highlight the importance of developing text-to-SQL methods based

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

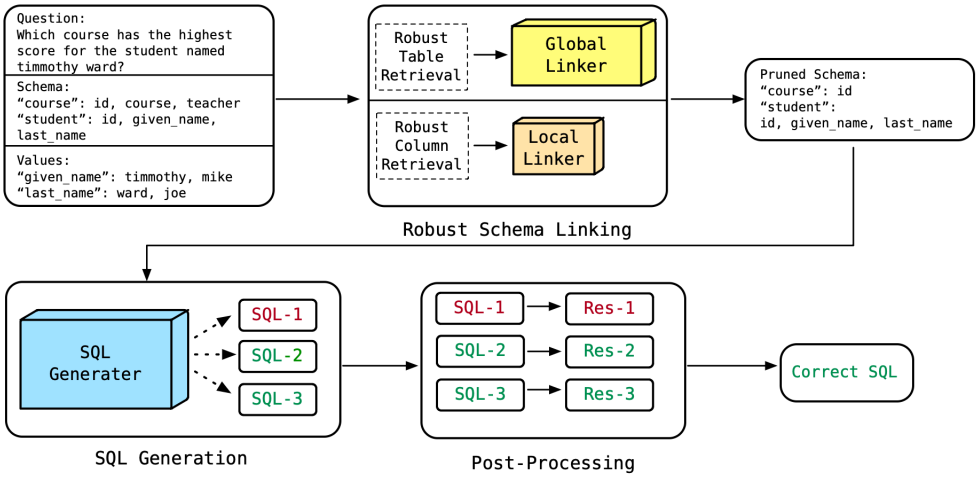

Figure 1: The Illustration of SageSQL pipeline.

on open-source LLMs, which enable cost-effective and privacy-preserving solutions for practical applications.

Despite many advantages offered by text-to-SQL methods based on open-source LLMs, they exhibit a significant performance gap compared to their closed-source counterparts. In particular, on the challenging BIRD benchmark[14], the popular open-source coding model StarCoder-15B[15] shows an approximately 25% lower execution accuracy than GPT-4, even the continue-pretrained and fine-tuned LLM that focuses on text-to-SQL, CodeS-15B[13], still lag-behind advanced in-context learning methods based on GPT-4o by 14%. This performance disparity underscores the limitations of current open-source LLMs in handling complex text-to-SQL tasks. To bridge this gap, exploring better text-to-SQL strategies with open-source LLMs is essential.

To enhance the capability of open-source LLMs in performing the text-to-SQL task, we propose SageSQL, a multi-agent framework for text-to-SQL. As illustrated in Figure 1, SageSQL generates SQL queries through a three-stage process. The first stage is schema linking, which prunes the large-scale database schema to retain only the schema components necessary to answer the given question. SageSQL employs a robust schema linking approach and enhances the reliability of this process by effectively avoiding overlooking both tables and columns. The second stage is diverse SQL generation, where multiple candidate SQL queries are generated with an ensemble method to achieve diversity. The third stage is post-processing. We employ a self-consistency approach to consolidate multiple diverse SQL queries into a single final SQL result.

2 Related Work

Text-to-SQL with Open-Source Models Before LLMs were widely used, researchers explored pretrained small-size language models with an ensemble framework to solve the text-to-SQL task. RAT-SQL[27] and LGE-SQL[2] encode database schemas and user questions with BERT[4] and utilize graph neural networks to model the foreign key relationship. With the growing capabilities of pretrained encoder-decoder language models, end-to-end translating approaches for text-to-SQL are employed. RASAT[20], PICARD[24], RESD-SQL[12] model the text-to-SQL task as a translation problem and make use of the T5[22] model to translate user questions into SQL queries. In later studies, large language models with transformer decoder architecture are gradually integrated into text-to-SQL solutions. ZeroNL2SQL[6] combines T5 models as SQL skeleton parsers and proprietary LLMs to obtain the complete SQL. DTS-SQL[19] designs a concise two-stage framework with two fine-tuned open-source LLMs, which first prune the database schema and then generate the SQL query with the pruned schema.

In-Context Learning Methods for Text-to-SQL Recently, many studies have focused on solving the text-to-SQL task using innovative prompting techniques and advanced text-to-SQL pipelines.

Most of these methods rely on the powerful capabilities of proprietary LLMs. For instance, ACT-SQL[32] generates chain-of-thought prompts automatically to improve text-to-SQL performance. DIN-SQL[18] decomposes complex problems into easily solvable problems and adds them to the prompt to enhance in-context learning. DAIL-SQL[7] samples semantically similar questions and SQL queries as prompts to further improve the prompt quality and gain a performance boost. PTD-SQL[16] proposes partitioning human-annotated examples in different banks and actively selects similar shots from various banks as examples at run-time. Other approaches concentrated on prompting LLMs multiple times, forming a multi-agent framework. For example, MAC-SQL[28] and DEA-SQL[30] designed integrated workflows to improve the results in various aspects. Additionally, [11] conducted sufficient experiments and summarized the pros and cons of each popular text-to-SQL method and proposed SuperSQL, which chooses the best option at different text-to-SQL stages with genetic algorithm.

Code Language Models After being trained on large-scale source code datasets, code language models demonstrate exceptional capabilities in code understanding, code generation, and program completion[3]. The interest in developing coding language models has been steadily increasing [15, 23, 26, 9] in the past few years. Due to the diverse range of training code languages, the relatively small amount of SQL code, and the constrained expressive capabilities of open-source models, the performance of open-source models on the text-to-SQL task is often suboptimal. As approaches to address this, CodeS[13] carried out continued pretraining on StarCoder[15] models with additional SQL data, and SENSE[10] proposes to fine-tune code language models with both strong and weak synthetic text-to-SQL data.

3 Preliminaries

Text-to-SQL Task. Let D be a relational database with schema $\mathcal{S}$. The database D consists of n tables $\{T_1, T_2, \ldots, T_n\}$, where each table T_i ($1 \leq i \leq n$) stores a set of contents denoted as V_i. Collectively, the contents of all tables are represented as $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$. <(Fixed) poor notation. which table? n is used twice> Given a user's natural language question Q, the task of text-to-SQL is to generate a SQL query sql such that the SQL query can extract the desired contents from D and answer the user's question.

In more practical scenarios, an optional natural language external knowledge E may be provided to guide the SQL generation. The external knowledge E can take various forms, such as domain-specific rules, special meanings of the database content, or additional metadata about the database. Incorporating all the definitions mentioned above, the text-to-SQL task can be formulated as follows:

$$sql = Parser(\mathcal{S}, \mathcal{V}, Q, E) \tag{1}$$

where $Parser$ denotes the text-to-SQL pipeline.

4 Robust Schema Linking

Schema linking serves as a preliminary stage that facilitates later SQL generation, yet according to [17], schema linking faces a challenge that simply trying to predict the sub-schema can cause performance degradation of text-to-SQL systems. This challenge arises from the fact that, though the schema linking stage filters out noisy tables and columns, it can bring an unignorable information loss. Even if a single required table or column is overlooked during the schema linking stage, the SQL query is guaranteed to fail, as every component of the SQL query is indispensable to correctly constructing the intended query.

Although schema linking has limitations, it remains an essential component for text-to-SQL solutions based on open-source models. This is because: (1) Open-source models have relatively smaller context windows, making it difficult to accommodate the entire database schema. (2) Unlike advanced proprietary LLMs like GPT-4, open-source models exhibit weaker comprehension capabilities for lengthy inputs, making it challenging to understand schemas and generate correct SQL simultaneously when presented with a large number of tokens. (3) In practical data warehouse scenarios, the number of tables and columns is often substantial, necessitating an efficient information retrieval process to maintain scalability.

Under these circumstances, text-to-SQL systems that are based on open-source models face a dilemma: directly predicting the sub-schema during the schema linking stage inevitably leads to information loss, and open-source models are still compelled to rely on this process due to their inherent limitations. Such a trade-off highlights a pressing question: How can we leverage schema linking effectively while minimizing issues caused by missing tables and columns? To solve the question, we propose a global-local approach to **avoid overlooking columns** in the schema linking stage. After that, we further refine our schema linking model through preference learning to **avoid overlooking tables**. This approach maximizes recall while maintaining high accuracy in schema linking, thus improving the robustness of the schema linking process.

4.1 Global-Local Schema Linking

As described in the previous section, schema linking is a challenging subtask that requires meticulous consideration of each table and column to ensure that no necessary table or column is overlooked. However, existing methods[19, 21, 8] perform schema linking in a in-one-go manner, which imposes high demands on the model's capabilities since the model needs to pay attention not only to the `JOIN` relationship of tables, but also to the datailed meaning of each column. As [17] pointed out, this type of schema linking approach leads to an unignorable performance degradation of text-to-SQL systems.

Algorithm 1 Global-Local Schema Linking

Require: Global schema linking model M_{global}, local schema linking model M_{local}, database schema $\mathcal{S}$, database value set $\mathcal{V}$, user question Q, external knowledge E
Ensure: Pruned database schema $\mathcal{S}_p$

$RV_{db} \leftarrow \texttt{GetRelatedValues}(\mathcal{V}, \mathcal{S}, Q, E)$
{Global schema linking}
$\mathcal{S}_{coarse} \leftarrow M_{global}(\mathcal{S}, RV_{db}, Q, E)$
$\mathcal{S}_p \leftarrow \mathcal{S}_{coarse}$
for all $T \in \mathcal{S}_{coarse}$ **do**
  {For each column in T}
  for all $C \in T$ **do**
    $RV_{col} \leftarrow \texttt{GetRelatedValues}(\mathcal{V}, C, Q, E)$
    {Local schema linking}
    if $M_{local}(T, RV_{col}, Q, E) == \texttt{True}$ **then**
      $\mathcal{S}_p \leftarrow \mathcal{S}_p \cup C$
    end if
  end for
end for
return $\mathcal{S}_p$ =0

To minimize loss of information, we introduce a **from-global-to-local** schema linking strategy, which introduces a local column scanning process to avoid overlooking required columns. As illustrated in Algorithm 1, we introduce two models to perform the schema linking task collaboratively. The global schema linking model M_{global} is a high-capacity model trained to perform coarse-grained selection of tables and columns. In particular, M_{global} is capable of identifying all tables necessary for constructing `JOIN` operations in cases where multi-table queries are needed. In contrast, the local schema linking model M_{local}, which is lightweight and fast, acts as a binary classification model to refine the results produced by M_{global}. Within the scope of the tables selected by M_{global}, M_{local} further examines unselected columns and their corresponding values to determine their relevance to the user question, thus identifying useful columns that may have been overlooked during the global linking stage.

4.2 Preference Learning for Schema Linking

Schema linking can be regarded as a retrieval task. This subtask emphasizes achieving a recall rate as high as possible while ensuring that precision remains sufficiently high, without overly prioritizing precision at the cost of recall. Previous schema linking approaches[19, 8] only used supervised fine-tuning for learning. However, limited model capacity often results in the model overlooking tables and columns during the schema linking stage. To address this, we adopt preference learning

to the global schema linking model M_{global} to instill in the model a bias that missing selections are unacceptable. By incorporating preference learning, we can effectively reduce the omission of tables in schema linking, thereby enhancing the overall robustness and reliability of our global-local schema linking framework.

Algorithm 2 Preference Data Augmentation

Input: Schema linking dataset D_{other}, column sample frequency K
Output: Schema linking preference dataset D_{dpo}
$D_{dpo} \leftarrow \varnothing$
{Augment each datapoint in D_{other}}
for all $(\mathcal{S}_i, Q_i, \mathcal{Y}_i) \in \mathcal{D}_{other}$ **do**
  $x_i \leftarrow \texttt{MakePrompt}(\mathcal{S}_i, Q_i)$
  {Acquire table augmentation data}
  for all $T \in \mathcal{S}_i$ **do**
    $y_{win} \leftarrow \texttt{Linearize}(\mathcal{Y}_i)$
    $y_{loss} \leftarrow \texttt{Linearize}(\mathcal{Y}_i \setminus T)$
    $D_{dpo} \leftarrow D_{dpo} \cup (x_i, y_{win}, y_{loss})$
  end for
  {Acquire column augmentation data}
  for $i = 1$ to K **do**
    $\mathcal{C} \leftarrow \texttt{RandomlySampleColumns}(\mathcal{Y}_i)$
    $y_{win} \leftarrow \texttt{Linearize}(\mathcal{Y}_i)$
    $y_{loss} \leftarrow \texttt{Linearize}(\mathcal{Y}_i \setminus \mathcal{C})$
    $D_{dpo} \leftarrow D_{dpo} \cup (x_i, y_{win}, y_{loss})$
  end for
end for

return D_{dpo} =0

During training, we have a pretrained LLM $M_{pretain}$ and the schema linking dataset $D = \{\mathcal{S}_i\, Q_i, \mathcal{Y}_i\}_{i=0}^{N}$, where $\mathcal{S}_i$ is the complete database schema, Q_i is the user question, and $\mathcal{Y}_i$ denotes the sub-schema that contains the exact tables and columns to answer Q_i. The database values and potential external knowledge are omitted for simplicity. We divide D into two parts D_{sft} and D_{other}, and D_{other} is further augmented by Algorithm 2 to obtain D_{dpo}. The `Linearize` function in Algorithm 2 transforms a schema into a token sequence. After having D_{sft} and D_{dpo}, we first supervised fine-tune the pretrained model $M_{pretrain}$ to obtain M_{sft}, and then tune M_{sft} using DPO to obtain M_{global}. The DPO process could be formulated as maximizing the following object:

$$\mathop{\mathbb{E}}_{(x,y_w,y_l)\sim D_{dpo}} \log \sigma \left(\beta \log \frac{p_\theta(y_w \mid x)}{p_{\text{ref}}(y_w \mid x)} - \beta \log \frac{p_\theta(y_l \mid x)}{p_{\text{ref}}(y_l \mid x)} \right) \tag{2}$$

where θ is the parameters of the language model and $p_\theta(y \mid x) = \prod_{t=1}^{T} p_\theta\left(y_t \mid y_{1:t-1}, x\right)$ is the conditional probability distribution of the predicted table name and the column name sequence given the prompt about the complete schema and the user question. T is the sequence length and t is the auto-regressive decoding step.

In summary, our schema linking approach achieved column robustness by employing the global-local schema linking strategy, and further achieved table robustness by adopting preference learning to the global schema linking model M_{global}. This combination ensures a more comprehensive and robust schema linking process across both tables and columns.

5 SQL Generation and Post-processing

5.1 Diverse SQL Generation

To enable the model to generate high-quality and diverse SQL queries, we fine-tuned the code LLMs and employed an ensemble sampling strategy during inference.

Pyramid Training Data. We performed data augmentation on the training data and fine-tuned our model using training datasets with three different schema scales. This approach was designed to enable the model to generate accurate SQL queries while also enhancing its ability to comprehend database schemas. The three schema scales in the training data are as follows:

- **Schemas just sufficient to answer the question**: These schemas help the model focus on learning the finer details of SQL generation. However, training exclusively on such data may cause the model to develop a bias toward using all available tables and columns.
- **Complete database schemas**: These schemas enable the model to learn how to identify and select the appropriate tables and columns from large-scale schemas to generate correct SQL queries.
- **Schemas with randomly added tables and columns beyond what is needed to answer the question**: These schemas allow the model to learn to appropriately ignore irrelevant noise and focus on using the most relevant tables and columns to generate SQL queries.

Ensemble Sampling. We observed that a well-aligned model tends to produce identical SQL queries with similar questions. To address this, we designed three different sampling methods and integrated them during inference to generate more diverse SQL queries.

- **Sampling with high temperature**: This method involves setting a high temperature at inference time, allowing the LLM to randomly explore and select different SQL generation paths.
- **Sampling with conditions**: By pre-conditioning the input with the keyword `WITH`, this approach forces the LLM to generate SQL queries using Common Table Expressions.
- **Beam search**: This technique employs multiple beams to represent different SQL keyword paths, enabling the generation of diverse SQL structures.

5.2 Postprocessing

In the post-processing phase, we adopted a self-consistency[29] approach. After generating diverse SQL results, each SQL query is pre-executed on the database to obtain its execution results. Erroneous and unreasonable results are filtered out, and the execution result that occurs most frequently is selected as the consistent result. Finally, one of the SQL queries leading to this consistent result is randomly chosen as the final output SQL.

6 Prompt Engineering

6.1 Linearized Table Representation

To provide enough information about the text-to-SQL question, we designed an integrated prompt structure as illustrated in Figure 2. The same structure is adopted in both the schema-linking module and the SQL generation module. Inspired by classical database design patterns, we incorporate different information needed for answering the user question into the prompt in a SQL data description language (DDL) style, which includes:

- **Table Defination** We follow the SQLite `CREATE TABLE` paradigm to indicate each table. Following the table definition are the column definitions, the primary key definition, and the foreign key relationships.
- **Column Attributes** To provide maximum information for the model, column types and optional `UNIQUE` and `NOT NULL` attributes are included in the prompt.
- **Column Comments** Since abbreviations and ambiguous columns are common in databases, we add column comments to describe the column additionally. Column comments include two types: (1) extended column name (for example, `amount` for column `amt` in Figure 2), (2) column meaning description (for example, `approved amount of the loan` for column `amt` in Figure 2).
- **Value Examples** After each table definition, value examples for each column are provided. For numeric columns, we randomly sample three values. For text-like columns including

`TEXT`, `DATE`, and `VARCHAR`, we extract three values stored in the database that have the highest similarity to the question embedding.

6.2 Value Retrieving

In the preprocessing stage, we utilize an embedding model to calculate the vector embeddings of each text-like value (`TEXT`, `DATE`, and `VARCHAR` etc.) in the database. At run-time, text-like values in the database with the highest similarity to the user question are retrieved. Although prior works [6, 13] propose to extract pertinent database content using ensemble or character-based approaches, we found that the light-weighted semantic similarity-based approach is already enough to extract the essential values for the text-to-SQL task. Moreover, unlike prior methods[19, 13], which first prune the database schema in an isolated manner and then extract database content to help generate SQL, we add value examples to the prompts both when pruning the schema and generating SQL. Such a strategy comes from the intuition that if a column contains values related to the current question, the column is more likely to be helpful to construct a SQL query to answer the natural language question. (For instance, if the user question is asking for a city, a column that has value `"New York"` is probably useful even if the column name or column description is an abbreviation or irrelevant)

7 Experiment

Model / Method	Spider		BIRD	
Name	**Dev-set**	**Test-set**	**Dev-set**	**Test-set**
Prompting Methods w/ Closed-Source LLMs				
GPT-4	72.9	-	49.2	54.9
C3-SQL[5] + ChatGPT	81.8	82.3	50.2	-
DIN-SQL[18] + GPT-4	82.8	85.3	50.7	55.9
DAIL-SQL[7] + GPT-4	83.5	86.2	54.8	57.4
TA-SQL[21] + GPT-4	85.0	-	56.2	59.1
PTD-SQL[16] + GPT-4	85.7	-	57.0	-
MAC-SQL[28] + GPT-4	86.8	82.8	57.6	59.6
SuperSQL[11] + GPT-4	87.0	-	58.5	62.7
E-SQL[1] + GPT-4o	-	-	65.6	66.4
Fine-tuning Open-Source LLMs				
SENSE-7B[10]	83.2	83.5	51.8	59.3
SENSE-13B[10]	84.1	86.6	55.5	63.4
SFT CodeS-7B[13]	85.5	-	55.8	60.3
SFT CodeS-15B[13]	85.4	-	57.2	59.2
Multi-Stage NL2SQL w/ Open-Source LLMs				
SageSQL-7B	**87.6**	–	**61.6**	–
SageSQL-32B	**88.5**	–	**70.2**	–

Table 1: Performance Comparison of Different Models

7.1 Evaluation Benchmarks

We evaluate the effectiveness of our method with recognized Text-to-SQL benchmarks across multiple datasets.

General Benchmark Spider[31] is a widely-recognized Text-to-SQL benchmark. Spider contains 7000 human-annotated Text-to-SQL pairs in its training set and 1034 pairs in the validation set, across 200 different databases and 138 domains.

Challenging Benchmark BIRD[14] is a challenging benchmark of large real-world databases. BIRD contains 95 databases across 37 fields and 9428 high-quality Text-to-SQL pairs. BIRD features massive and dirty database contents and requires Text-to-SQL systems to reason on external expert knowledge to generate SQL queries.

7.2 Evaluation Metric

We report the common evaluation metric: Execution Accuracy (EX). EX determines equivalence between a predicted SQL query and a reference SQL query if they produce identical results across various database instances. EX is considered an accurate measurement of Text-to-SQL methods since multiple correct SQL queries can differ in output style.

7.3 Main Evaluation Results

As illustrated in Table1, SageSQL achieved an execution accuracy (EX) of 87.6% on the Spider-dev dataset and 61.6% on the BIRD-dev dataset when using the 7B model. With the 32B model, SageSQL further improved its performance, achieving 88.5% EX on Spider-dev and 70.2% EX on BIRD-dev, surpassing a range of state-of-the-art (SOTA) methods. Notably, the baseline SOTA methods primarily rely on the GPT-4 model, which entails significant computational costs and poses potential privacy risks. This performance improvement demonstrates that by leveraging a multi-agent framework and open-source code models, it is possible to achieve efficient text-to-SQL processing while maintaining low costs.(For Spider-test and BIRD-test, the models need to be submitted to the organizers to obtain results. Due to time constraints, we have not included the corresponding results here but will attempt to provide them in the coming weeks.)

7.4 Rubust Schema Linking Evalution

We further evaluated the Robust Schema Linking module by analyzing its performance on the BIRD-dev dataset, focusing on the number of test cases successfully recalled. Specifically, for the 1,534 test cases in BIRD-dev, we defined two types of negative impacts caused by schema linking: (1) Table Loss: At least one table is omitted during the schema linking process. (2) Column Loss: While all tables are correctly identified, at least one column is omitted. These two types of negative impacts are mutually exclusive. For each test case, if either type of negative impact occurs, the test case is determined to fail and the less loss occur is considered the better.

Table 2: Robust Schema Linking Evaluation

	# Table Loss	**# Column Loss**	**# All Loss**
Global(SFT) Schema Linking	166	143	309
Global(SFT+DPO) Schema Linking	152	96	248
Global(SFT) + Local Schema Linking	166	72	238
Global(SFT+DPO) + Local Schema Linking	**152**	**47**	**199**

As shown in Table 2, the simple global schema linking method fails on 309 out of 1534 data points in the BIRD-dev dataset. By employing a combined global and local schema linking approach, we significantly reduced the number of failed data points by minimizing column loss. Furthermore, we fine-tuned the global schema linking model using DPO, which further reduced the table loss. Ultimately, the integration of the global-local schema linking method with DPO resulted in reductions in both table loss and column loss, leaving only 199 failed data points and achieving the best overall performance.

7.5 Ablation Study

We conducted ablation study as illustrated in Table 3. We conducted ablation experiments by removing the schema linking module, replacing the ensemble SQL generation process with simple

Table 3: Ablation Study on SageSQL modules

Module Name	EXecution Accuracy on BIRD-dev
SageSQL-32B	70.2
w/o Schema Linking	64.5 (-5.7)
w/o Diverse SQL Generation	68.2 (-2.0)
w/o Self-Consistency	67.8 (-2.4)

multiple sampling, and eliminating the self-consistency post-processing step. The results showed varying degrees of decline in the EX of SageSQL, with the removal of the schema linking module leading to the most significant performance degradation. This further highlights the critical role of schema linking in enhancing the performance of open-source models on text-to-SQL tasks.

7.6 Discussion about Post-processing

The conventional evaluation methods for text-to-SQL tasks typically consider only a single predicted SQL query to determine correctness. In this work, we extend the discussion to scenarios where multiple SQL queries are generated, and at least one is correct. Specifically, we employ ensemble sampling to generate 24/36 SQL queries and evaluate the probability of producing a correct SQL query among the outputs.

Table 4: BIRD-dev evaluation (Top-N) scenario

	EXecution Accuracy on BIRD-dev
SageSQL-32B (Top-1 with Self-Consistency)	70.2
SageSQL-32B (Top-24)	79.2
SageSQL-32B (Top-36)	81.5
w/o Diverse SQL Generation (Top-24)	74.6

As shown in Table 4. We observed that when generating 24 SQL queries, as many as 79.2% of the data points contained at least one correct SQL query. However, the Self-Consistency approach, which selects a single SQL query as the final output, resulted in only 70.2% of the data points achieving the correct SQL. This indicates that there is room for improvement in designing a more effective selection module to identify the best SQL query from multiple candidates. Furthermore, we observed that our Diverse SQL Generation module significantly expands the SQL search space. Without employing the Diverse SQL Generation approach, up to 5% of the data points among the 24 generated SQL queries would fail to include a correct SQL query. This reduction decreases the likelihood of achieving the correct SQL result.

8 Conclusion

This paper introduces SageSQL, a multi-agent framework designed to enhance the performance of open-source LLMs in text-to-SQL tasks. By addressing the inherent limitations of open-source models through innovations such as robust schema linking, diverse SQL generation, and self-consistency-based post-processing, SageSQL achieves state-of-the-art results among open-source solutions and narrows the performance gap with proprietary models. The proposed global-local schema linking and preference learning mechanisms ensure minimal loss of critical information, while the diverse SQL generation module encourages structural variety in outputs, thereby improving execution accuracy on challenging datasets like BIRD.

Our experimental results validate the efficacy of SageSQL, showing significant improvements over existing methods in both general and challenging benchmarks. Moreover, the cost-efficiency and privacy advantages of using open-source LLMs position SageSQL as a practical alternative for real-world applications, especially in environments where proprietary solutions are infeasible due to financial or regulatory constraints. Future work will explore the integration of external knowledge bases and further optimization of the multi-agent framework to enhance scalability and adaptability across diverse database systems.

References

[1] Hasan Alp Caferoğlu and Özgür Ulusoy. E-sql: Direct schema linking via question enrichment in text-to-sql. *arXiv preprint arXiv:2409.16751*, 2024.

[2] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. In *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)*, pages 2541–2555, 2021.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Henrique Ponde de Oliveira Pinto Qiming Yuan, Jared Kaplan, and et.al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

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

[6] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. Combining small language models and large language models for zero-shot nl2sql. *Proc. VLDB Endow.*, 17(11):2750–2763, 2024.

[7] 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. *Proc. VLDB Endow.*, 17(5):1132–1145, 2024.

[8] Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, and Rasa Hosseinzadeh. Msc-sql: Multi-sample critiquing small language models for text-to-sql translation. *arXiv preprint arXiv:2410.12916*, 2024.

[9] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, and et.al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[10] Yang Jiaxi, Hui Binyuan, Yang Min, Yang Jian, Lin Junyang, and Zhou Chang. Synthesizing text-to-SQL data from weak and strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875, 2024.

[11] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. The dawn of natural language to sql: Are we fully ready? *Proc. VLDB Endow.*, 17(11):3318–3331, 2024.

[12] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, 2023.

[13] 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. *Proc. ACM Manag. Data*, 2(3), 2024.

[14] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2024.

[15] Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, and et.al. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023.

[16] Ruilin Luo, Liyuan Wang, Binghuai Lin, Zicheng Lin, and Yujiu Yang. Ptd-sql: Partitioning and targeted drilling with llms in text-to-sql. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3767–3799, 2024.

[17] Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The death of schema linking? text-to-sql in the age of well-reasoned language models. *arXiv preprint arXiv:2408.07702*, 2024.

[18] Mohammadreza Pourreza and Davood Rafiei. Din-sql: decomposed in-context learning of text-to-sql with self-correction. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2024.

[19] Mohammadreza Pourreza and Davood Rafiei. Dts-sql: Decomposed text-to-sql with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, 2024.

[20] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. RASAT: Integrating relational structures into pretrained Seq2Seq model for text-to-SQL. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3215–3229, 2022.

[21] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-SQL generation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 5456–5471, 2024.

[22] 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.

[23] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, and et.al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[24] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, 2021.

[25] Michael Stonebraker and Andrew Pavlo. What goes around comes around... and around... *SIGMOD Record.*, 53(2):21–37, 2024.

[26] CodeGemma Team. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*, 2024.

[27] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, 2020.

[28] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. Mac-sql: A multi-agent collaborative framework for text-to-sql. *arXiv preprint arXiv:2312.11242*, 2023.

[29] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[30] Yuanzhen Xie, Xinzhou Jin, Tao Xie, Mingxiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, Chengxiang Zhuo, Bo Hu, and Zang Li. Decomposition for enhancing attention: Improving LLM-based text-to-SQL through workflow paradigm. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10796–10816, 2024.

[31] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018.

[32] Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3501–3532, 2023.

A Prompt Engineering Example

Question & Schema representation for text-to-SQL

```
[Question]
How many loans start from 1994 are borrowed?
[External Knowledge]
"Loans are borrowed" refers to: status is B
[Schema]
TABLE "loan" (
"loan_id" INT,
    -- (ientifying the loan data)
"account_id" INT NOT NULL,
    -- (number identifying the account)
"date" DATE NOT NULL,
    -- (the date when the loan is approved)
"amt" INT NOT NULL,
    -- amount (approved amount of the loan)
"ln_sts" TEXT NOT NULL,
    -- loan status (repayment status)
PRIMARY KEY ("loan_id"),
FOREIGN KEY ("account_id")
    REFERENCES "account"("account_id")
);
/* Value examples for each column:
"loan_id": 4959, 4961, 4962
"account_id": 2, 19, 25
"date": 1994-01-05, 1996-04-29, 1997-12-08
"amt": 80952, 30276, 318480
"ln_sts": 'B', 'D', 'A'
*/
(...other tables are omitted...)
```

Figure 2: An exemplary prompt for the text-to-SQL question. We use the same structure in different text-to-SQL stages. Here, we show the prompt in the SQL generation stage.