

SDE-SQL: Enhancing Text-to-SQL Generation in Large Language Models via Self-Driven Exploration with SQL Probes

Wenxuan Xie^{1,2*}, Yaxun Dai⁴, Wenhao Jiang^{3†}

¹Fudan University, Shanghai, China

²Shanghai Innovation Institute, Shanghai, China

³Guangdong Laboratory of AI and Digital Economy (SZ), Shenzhen, China

⁴Soochow University, Suzhou, China

wxxie25@m.fudan.edu.cn, cswjiang@gmail.com

Abstract

Recent advances in large language models (LLMs) have led to substantial progress on the Text-to-SQL task. However, existing approaches typically depend on static, pre-processed database information supplied at inference time, which restricts the model’s capacity to deeply comprehend the underlying database content. In the absence of dynamic interaction, LLMs are limited to fixed, human-curated context and lack the ability to autonomously query or explore the data. To overcome this limitation, we introduce **SDE-SQL**, a novel framework that empowers LLMs to perform **Self-Driven Exploration** of databases during inference. This is achieved through the generation and execution of **SQL probes**, enabling the model to actively retrieve information and iteratively refine its understanding of the database. Unlike prior methods, **SDE-SQL** operates in a **zero-shot** setting, requiring no in-context demonstrations or question-SQL pairs. Evaluated on the BIRD benchmark with Qwen2.5-72B-Instruct, **SDE-SQL** achieves an **8.02%** relative improvement in execution accuracy over the vanilla Qwen2.5-72B-Instruct baseline, establishing a new state-of-the-art among open-source methods without supervised fine-tuning (SFT) or model ensembling. Furthermore, when combined with SFT, **SDE-SQL** delivers an additional **0.52%** performance gain. Our code is publicly available at <https://github.com/Lancelot-Xie/SDE-SQL>.

1 Introduction

Text-to-SQL is a long-standing task in natural language processing that focuses on translating natural language questions into executable SQL queries. This capability not only empowers non-expert users to interact with structured databases seamlessly,

but also mitigates hallucination issues in question-answering systems by grounding responses in factual, database-stored information.

Recent advances in large language models (LLMs) have led to significant improvements in the performance and accuracy of Text-to-SQL systems. LLM-based approaches have surpassed **90%** execution accuracy on the original Spider dataset (Yu et al., 2018), and have demonstrated promising results on more complex and diverse benchmarks such as BIRD (Li et al., 2023). Despite these advances, a noticeable gap remains between current model performance and human-level capabilities—particularly on the recently introduced Spider 2.0 benchmark (Lei et al., 2025), which poses more realistic and challenging scenarios for semantic parsing. Contemporary large language model (LLM)-based approaches to Text-to-SQL typically comprise three core components: schema linking, SQL generation, and SQL refinement. In the schema linking stage, prior work has primarily focused on aligning natural language questions with relevant database schema elements, improving precision and contextual relevance. During SQL generation, various methods have been proposed to decompose complex questions and incorporate reasoning strategies. Recent work has also explored incorporating execution feedback directly into the reasoning and generation process, with ReEx-SQL interleaving intermediate SQL execution with decoding and learning execution-aware reasoning through reinforcement learning (Dai et al., 2025a). In the refinement stage, the categorization of SQL error types has become more systematic, enabling the development of targeted correction mechanisms.

Despite these advances, one crucial aspect of SQL remains largely underexplored: its inherent interactivity as a database interface that supports fast and informative execution. This underutilized property may partially account for the performance

*Work done during an internship at the Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ).

†Corresponding author.

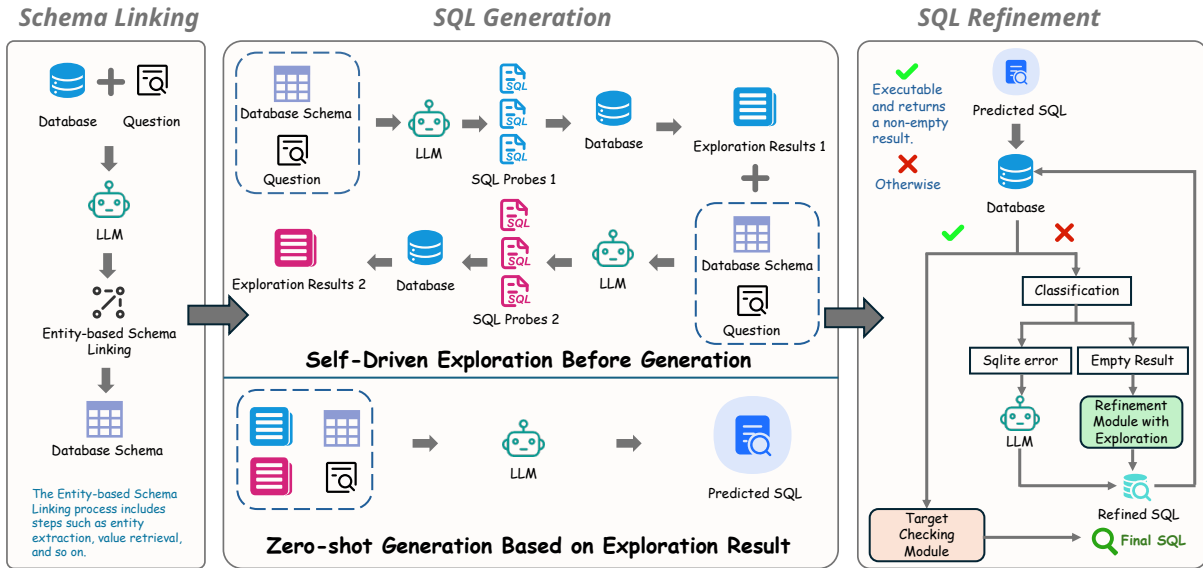


Figure 1: The Workflow of SDE-SQL, which consists of three parts: 1) Schema Linking: which retrieves and selects useful database schema; 2)SQL Generation: performing zero-shot SQL generation based on two-phase self-driven exploration; 3)SQL Refinement: which refines the SQL with the execution results of the Sub-SQLs and SQL Probes.

gap between LLM-based systems and human experts.

To address this, we propose **SDE-SQL**, a novel framework that incorporates **Self-Driven Exploration** into both the generation and refinement stages, as illustrated in Figure 1. In addition to generating the final SQL query that directly answers the natural language question, the model autonomously generates and executes a sequence of auxiliary queries—termed **SQL Probes**—designed specifically to explore and extract informative signals from the database.

For schema linking, we leverage entity-based techniques including value retrieval and soft linking. During the generation phase, the model engages in a two-stage exploration process based on the question and schema, enabling it to iteratively refine its understanding of the database content and perform zero-shot reasoning grounded in the retrieved information.

Following generation, we incorporate a two-stage exploration process into the refinement phase. For SQL queries that return explicit execution errors, the model directly revises them based on the error feedback. For SQL queries that execute successfully but return empty results, the failure may arise from either problematic user questions (e.g., ambiguous or unanswerable concepts/values) or logically flawed executable SQL, such as faulty join conditions or mismatched filter values with respect to the actual database content (Wang et al.,

2023; Dong et al., 2025; Wang et al., 2025; Ghosh et al., 2025). The first stage of exploration uses the execution results of decomposed sub-queries (*Sub-SQLs*) to help the model diagnose the underlying issue. In the second stage, the model generates targeted **SQL Probes** to explore possible solutions, and selects the most promising one to produce the final refined query.

Empirically, **SDE-SQL** achieves an execution accuracy of **67.67%** on the BIRD benchmark using Qwen2.5-72B-Instruct (Yang et al., 2024) in a zero-shot setting. With supervised fine-tuning (SFT), the performance further improves to **68.19%**.

Our main contributions are as follows:

- We propose **SDE-SQL**, a novel framework that leverages **Self-Driven Exploration** to enhance the reasoning and interaction capabilities of LLMs in the Text-to-SQL task, significantly narrowing the gap with human experts.
- We introduce a unified exploration mechanism across both SQL generation and refinement stages, enabling LLMs to actively query the database, diagnose potential errors, and iteratively improve query quality.
- We conduct extensive experiments on the BIRD and Spider benchmarks, along with ablation studies, validating the effectiveness of Self-Driven Exploration.

- We build a small-scale dataset for supervised fine-tuning (SFT) on exploration and generation tasks, and show that targeted module-level fine-tuning further improves the performance of SDE-SQL.

2 Related Work

Transforming natural language questions into database queries is a classic task, the earliest works used inductive logic programming and human-designed templates to accomplish this task (Zelle and Mooney, 1996). In recent years, the advancement of Text-to-SQL technologies can be broadly categorized into two stages, driven by progress in natural language processing.

2.1 Traditional Seq2Seq Model-Based Methods

Previous work primarily focused on improving encoding or decoding methods, as the seq2seq model framework consists of two main components, the encoder and the decoder. IRNet employed a bidirectional LSTM to encode the question and a self-attention mechanism to encode the database schema, ultimately using an LSTM as a grammar-based decoder (Guo et al., 2019). In order to effectively capture the relationship between the database schema and the question, RAT-SQL develops an encoder with a relation-aware self-attention mechanism (Wang et al., 2020). After that, Cai et al. (2021) and Cao et al. (2021) utilized graph neural networks to encode the relationships between the schema and the query. Leveraging the exceptional capabilities of pre-trained language models (PLMs) across various NLP tasks, Hwang et al. (2019) was the first to incorporate BERT as its encoder. For improvements in the decoder, Xu et al. (2017) and Choi et al. (2021) focused on sketch-based decoding method. To reduce time consumption during inference, SDSQL presented the Schema Dependency Learning and removed execution-guided (EG) decoding strategy (Hui et al., 2021).

2.2 LLM-Based Methods

With the advent of LLMs, the Text-to-SQL field has experienced a groundbreaking innovation, bringing about significant changes in the approach to the task.

Methods Based on Prompt Engineering Rajkumar et al. (2022) evaluated the potential of LLMs

in the Text-to-SQL task, demonstrating the remarkable capability of LLMs in this task. Building on in-context learning, DAIL-SQL (Gao et al., 2024) introduced a novel prompt engineering approach that improves the Text-to-SQL performance of LLMs through question representation, demonstration selection, and demonstration organization. Based on Chain-of-Thought (CoT) reasoning style (Wei et al., 2022), DIN-SQL (Pourreza and Rafiei, 2023), Divide-and-Prompt (Liu and Tan, 2023), CoE-SQL (Zhang et al., 2024a) and SQLfuse (Zhang et al., 2024b) designed CoT templates with reasoning steps in the prompt to elicit chain thinking. To enhance the ability of LLMs in handling complex problems, QDecomp (Tai et al., 2023), DIN-SQL (Pourreza and Rafiei, 2023), MAC-SQL (Wang et al., 2025) and MAG-SQL (Xie et al., 2024) decomposed complex natural language questions and solve them step by step. Besides, MCS-SQL (Lee et al., 2025), CHASE-SQL (Pourreza et al., 2025) and CHESS (Talaie et al., 2024) enhanced performance by generating a large set of candidate SQL queries during the inference stage and selecting the most suitable ones.

Methods Based on Fine-tuning Although prompt engineering methods based on closed-source models, like GPT-4o (OpenAI, 2024), perform well in the Text-to-SQL task, they face issues such as high costs, inability to guarantee privacy, and limited flexibility. Therefore, fine-tuning open-source models for the Text-to-SQL task holds significant practical value and application potential. DTS-SQL (Pourreza and Rafiei, 2024) and SQLfuse (Zhang et al., 2024b) explored fine-tuning LLMs for both schema linking and SQL generation. SQL-PaLM (Sun et al., 2024), Open-SQL (Chen et al., 2024), XiYan-SQL (Liu et al., 2026) and CodeS (Li et al., 2024) fine-tuned open-source LLMs on carefully selected data, while CodeS specifically adopted an incremental pre-training approach using a specially curated SQL-centric corpus. In addition, there are some novel perspectives. DELLMHong et al. (2024) specifically fine-tuned a Data Expert Language Model that provides domain knowledge, while SQL-GEN Pourreza et al. (2024) proposed a novel Mixture-of-Experts (MoE) architecture to handle multiple SQL dialects.

Despite these advances, existing methods mainly improve Text-to-SQL through better prompting, candidate selection, or model fine-tuning, while database execution is often used only as auxiliary

feedback for reasoning or correction. In contrast, our method explicitly introduces self-driven exploration as a core inference mechanism, where the model actively issues SQL Probes to interact with the database, gather evidence, and support both SQL generation and refinement.

3 Methodology

3.1 Entity-based Schema Linking

In the Text-to-SQL task, schema linking refers to the process of identifying and selecting the relevant tables, columns, and values from the database based on the input natural language question. To improve the accuracy of linking, we use an entity-based linking approach, including **Value Retrieval** and **Soft Schema Linking**.

3.1.1 Entity-based Value Retrieval

Similar to the retrieval module in Talaei et al. (2024), we first employ an LLM to extract entities from the natural language question through few-shot learning. And then the value retriever identifies similar values in the database based on Locality Sensitive Hashing (LSH) and semantic similarity.

3.1.2 Entity-based Soft Schema Linking

To improve the tolerance in the schema linking stage, we chose the soft schema linking method, like the approach in Xie et al. (2024). We employ a one-shot manner to prompt LLM to select the relevant columns based on each entity. For the selected columns, we provide as much detailed information as possible during the subsequent SQL generation, including the column name, type, column description, value examples, and value descriptions. For the unselected columns, we only retain the column name and type. This approach not only significantly reduces the input length, allowing the language model to focus on the most relevant database schema during generation, but also enhances tolerance by preventing the removal of useful columns that were not chosen.

3.2 Generation Based on Self-Driven Exploration

In previous Text-to-SQL research, SQL has often been viewed primarily as an intermediate result or final output, with its inherent functionality mostly overlooked. Therefore, we introduce the concept of **SQL Probes**. SQL Probes, literally meaning

SQL queries that function as probes, are specifically designed for exploring the database based on current natural language question. Formally, we define the task as a mapping from a natural language query Q and a database schema D to a corresponding SQL query S . The natural language query Q is composed of two parts: the **target** and the **conditions** (Xie et al., 2024). Typically, the target corresponds to the main SELECT clause in the SQL query S , while the conditions correspond to the other clauses in S , such as the WHERE clause. Figure 2 is an example.

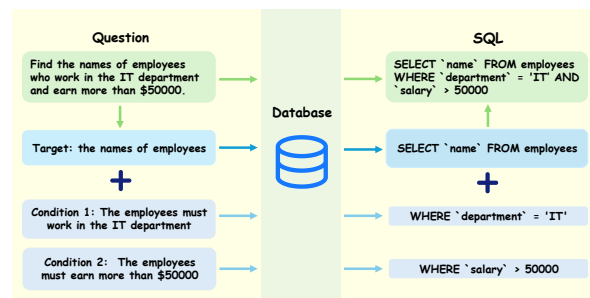


Figure 2: An example of Text-to-SQL.

To obtain a specific SQL representation, entities must first be mapped to the corresponding columns and values in the database. Whether this step can be executed accurately depends on how well the language model understands the database.

However, the information provided by the previously processed database schema is far from sufficient. Real-world databases are often highly complex and messy. Different tables may contain many columns with the same meaning (representing the same item), and the values in these columns might have different formats, with some values even existing only in specific tables. In the absence of sufficient information, LLM can only randomly identify combinations from these similar columns and values. This is also one of the key reasons behind the LLM’s especially unstable performance in this task. During evaluation, it is frequently observed that the model can correctly predict some examples at times, while failing on the same examples at other times. In prior work, some methods have involved generating multiple SQL-candidates with the language model, followed by selecting the most appropriate one. Nevertheless, this approach fails to address the underlying problem. We propose that the most fundamental solution is to empower LLM with the ability to dynamically interact with the database. In SDE-SQL, LLM performs a two-

stage self-driven exploration within the database before generation.

3.2.1 Candidates Exploration

The goal of this stage of exploration is to enable the large language model to query the database for information regarding both the Targets and a single Condition, and then select appropriate candidates for each target and condition. Since an entity in a natural language question is mapped to either a column or a value (or both a column and a value) in the database, LLM needs to determine the candidate columns and candidate values for each entity. Initially, the language model generates several **Base SQL Probes**, which enumerate candidate columns for the Targets. These SQLs focus solely on querying the Targets without any additional conditions. Following this, Condition SQL Probes are created, where each Probe typically extends a Base SQL Probe by adding a column candidate and maybe a value candidate corresponding to a specific condition. Assuming each set of candidates contains two options, the generation of Condition SQL Probes is illustrated in Figure 3, where each root-to-leaf path corresponds to a specific Condition SQL Probe. We refer to the condition description of each Condition SQL Probe as a **Condition Description Candidate**. For example, in Figure 3, one Condition Description Candidate is:

```
SELECT Phone FROM users WHERE name =  
'John_Charlie_Hinton' AND location = '  
United_Kingdom';
```

3.2.2 Combinations Exploration

Based on the results of the previous stage’s exploration, the scope of candidates has been narrowed down. Now, it is necessary to combine all the conditions to find the most suitable candidate combination. For SQL queries that return no results, the corresponding candidate combination is definitely unsuitable.

3.2.3 Zero-shot Generation with Exploration Results

In our experiments, we found that existing methods do not fully leverage the large language model’s potential for SQL generation. For example, strategies such as designing new decomposition approaches to allow the model to progressively solve complex problems, using various prompt techniques to generate multiple candidates for selection, or employing search strategies like Monte Carlo tree

search(MCTS) to enhance the inference capability of language models, can lead to modest improvements in model performance. However, these gains are still significantly smaller than those achieved by providing the model with sufficient information.

Therefore, in SDE-SQL, the LLM generator generates SQL based on the database schema and the results from the previous two exploration stages, without relying on any question-SQL pairs as few-shot examples or using any question decomposition strategies. To improve the accuracy and robustness of SQL generation, we adopt a self-consistency strategy that selects the most consistent answer by comparing the execution results of multiple generated SQL queries.

3.3 Refinement Based on Self-Driven Exploration

In the past, existing techniques based on In-Context Learning have introduced detection and repair solutions for Text-to-SQL errors, with each solution differing in its approach to error identification algorithms and the supplementary data provided to assist LLM in comprehending and rectifying these errors.(Shen et al., 2025)

For Syntax errors and Schema errors, the error feedback after execution already contains sufficient information, allowing LLMs to effectively complete the correction of SQL. However, for some other more complex errors, they typically result in empty query results without any error messages. Even when humans attempt to correct these errors, they cannot do so in one go; instead, they need to write some SQL statements for debugging and diagnose the problem based on the execution results of these queries. The current approach involves continuously regenerating until the repair is successful or the attempt limit is reached. Throughout this repair process, LLM does not receive any useful information, and its reasoning abilities are not fully utilized. In other words, the reason for the error is never identified.

Therefore, in **SDE-SQL**, we introduce a comprehensive **Self-Driven Exploration** phase prior to SQL revision. For queries that yield empty results, the refinement process is divided into three distinct stages: the **Error Cause Identification Stage**, the **Solution Exploration Stage**, and the **Modification Stage**, as illustrated in Figure 4.

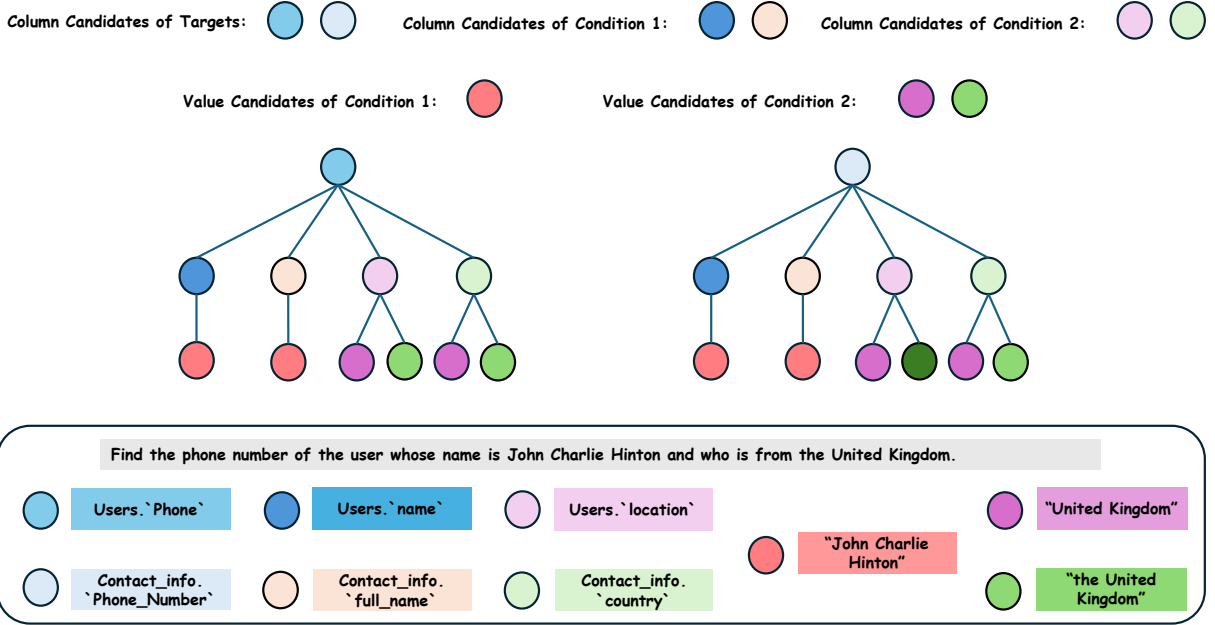


Figure 3: Condition SQL Probes Generation Process Illustrated Using a Tree Structure.

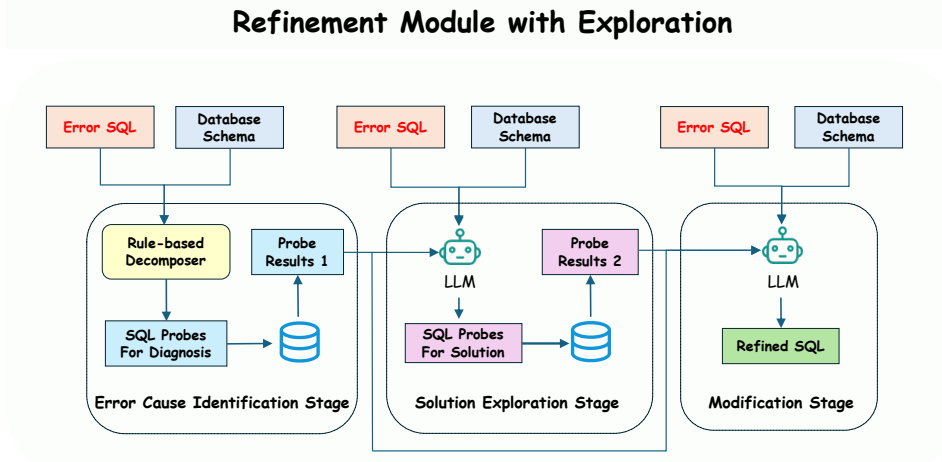


Figure 4: An illustration of the proposed refinement process with exploration in SDE-SQL.

3.3.1 Error Cause Identification Stage

In a complex SQL statement, multiple tables may be involved and multiple conditions may be applied simultaneously, making it difficult to pinpoint the issue by directly analyzing the entire SQL. Therefore, we need to conduct a fine-grained diagnosis. To generate a series of Sub-SQLs as SQL Probes for diagnostic purposes, we developed a decomposer based on **SQLglot**. The decomposer first converts complex SQL queries into **Abstract Syntax Trees (ASTs)** and then identifies indivisible condition units by analyzing node types and their relationships. These identified subtrees within the AST serve as the foundation for generating semantically valid Sub-SQLs, and the execution results of these Sub-SQLs will be provided to LLM to

assist it in accurately diagnosing and pinpointing issues in the original query. AST-based SQL parsing has also been used in prior work for SQL decomposition (Dai et al., 2025b). An example of the decomposition result is shown in Figure 8.

3.3.2 Solution Exploration Stage

We have summarized five possible reasons that may lead to an empty query result. At this stage, LLM needs to analyze the probe results in the previous stage to derive hypotheses about possible error causes, and then generate a series of SQL probes to assist in exploring potential solutions to these errors.

Conditions conflict or condition duplication

This error refers to situations where data can be

found when executed under a single condition, but when multiple conditions are combined, no data that meets the requirements can be found (resulting in an empty query result). There are two possible reasons for this error: conflicting combinations of multiple different conditions or redundant descriptions of a single condition using different columns. (i) Conditions conflict typically arises when an entity in a condition corresponds to multiple possible candidate columns, and only a specific candidate column combined with other conditions can yield the corresponding data item. An example is shown in Figure 5. (ii) Condition duplication occurs when an entity in a condition maps to multiple candidate columns, causing the SQL generated by the large language model to inadvertently employ these various candidate columns in describing the same condition, ultimately resulting in the failure to retrieve data that fulfills the intended condition.

Unnecessary Table Joins The SQL may include unnecessary table joins, resulting in no records satisfying the conditions in the final intersection.

Mismatch between column and value This error arises when either the value format does not match the selected (but correct) column, or when a similar-looking column is chosen that does not contain the intended value.

Sub-query Scope Inconsistency Sometimes, the scope of the sub-query may be inconsistent with that of the main query, especially when using the MIN/MAX functions in the sub-query, which often leads to an empty query result. Figure 6 shows an example that the row of data retrieved in the sub-query does not exist in the result after the JOIN of these two tables.

3.3.3 Target Checking After Refinement

For an SQL query, the most important part is actually the target of the query, which refers to the columns being selected in the SELECT clause. If the query target in SQL does not align with the original query target in the natural language question, then the transformation is undoubtedly a failure. However, when LLMs generate SQL, they sometimes include columns that are not part of the query target in the SELECT clause, such as columns used in the conditions. Therefore, after refining the SQL based on the execution results, it is necessary to check whether the query target in the SQL matches the query target in the natural language question. To

avoid introducing new errors at this stage, we allow the large language model to only determine if unnecessary target columns are selected in the SQL. If such columns are found, they will be removed without affecting the execution. The procedure is illustrated in Figure 7.

3.4 Supervised Fine-Tuning (SFT)

To further enhance the model’s ability to autonomously explore the database and utilize exploration results to generate more accurate SQL, we also perform supervised fine-tuning (SFT) on the model. The training data is sampled from the training set of BIRD with Qwen2.5-72B-Instruct. We employed a prompt-based pipeline to roll out data, and the examples that eventually produced correct SQL were regarded as valid data for fine-tuning the model.

Among the 9,428 data points, 5,231 valid samples were obtained through sampling. From the reasoning trajectory of each example, we extract two components: (i) the exploration phase, where SQL probes are generated; and (ii) the prediction phase, where the final SQL query is generated based on the exploration results.

4 Experiments

In this section, we first introduce the experimental setup, and then report and analyze the results.

4.1 Experimental Setup

4.1.1 Dataset and Metrics

Spider (Yu et al., 2018) is a widely adopted benchmark dataset for the Text-to-SQL task. It is large-scale, cross-domain, and complex, containing 10,181 natural language questions and 5,693 corresponding SQL queries across 200 different databases. As a challenging benchmark of Text-to-SQL task, the recently proposed BIRD dataset (Li et al., 2023) includes 95 large-scale real databases with dirty values, featuring 12,751 unique question-SQL pairs. The databases within the BIRD dataset, similar to those in real-world scenarios, exhibit inherent ambiguities. Accordingly, detailed descriptions are provided for each column, along with external knowledge. In this work, we choose Execution Accuracy (EX) as the metric, since it reflects the accuracy of the results returned by the executed SQL queries. This metric considers various SQL formulations for the same question, providing a more precise and fair evaluation of the

Method	Simple	Moderate	Challenging	All
SDE-SQL + Qwen2.5-72B-Instruct	74.92	57.76	53.10	67.67
w/o Soft Schema Linker	73.51	58.84	50.34	66.88 _{↓0.79}
w/o Exploration Before Generation	72.97	56.46	48.97	65.71 _{↓1.96}
w/o Refinement Module	72.97	55.60	48.97	65.45 _{↓2.22}
w/o Exploration in Refinement	72.86	56.68	51.72	65.97 _{↓1.70}
w/o Target Checking	73.19	57.76	49.66	66.30 _{↓1.37}
w/o Exploration in Generation & Refinement	72.11	54.31	48.28	64.47 _{↓3.20}
SDE-SQL + Fine-tuned Explorer	74.16	59.26	55.17	67.86 _{↑0.19}
SDE-SQL + Fine-tuned Generator	74.49	58.19	55.86	67.80 _{↑0.13}
SDE-SQL + Fine-tuned Explorer & Generator	74.70	58.84	56.55	68.19 _{↑0.52}

Table 1: Execution accuracy of SDE-SQL on BIRD dev set in the ablation study.

outcomes. Due to occasional runtime variability under timeout-constrained execution, we run the execution-based evaluation script 5 times and report the average results. In each run, every predicted SQL is judged by a single execution.

4.1.2 SFT Settings

For both exploration task and generation task, we conducted 24-hour training on 8 NVIDIA A800 GPUs with Qwen2.5-72B-Instruct. The detailed training hyperparameters are provided in Table 5.

4.1.3 Baselines

To enable a comprehensive comparison, we selected representative methods based on **closed-source** models and representative methods based on **open-source** models **without model ensemble** as baselines.

4.2 Main Results

4.2.1 BIRD Results

When evaluated on the BIRD dev dataset, **SDE-SQL** based on Qwen2.5-72B-Instruct outperforms **most GPT-4-based methods** and the majority of open-source models, achieving an execution accuracy of **68.19%** after fine-tuning, as shown in Table 2. Even in the training-free setting, it achieves a strong performance of **67.67%**, further highlighting the effectiveness of our approach.

4.2.2 Spider Results

As shown in Table 3, SDE-SQL fine-tuned solely on the BIRD training set achieves competitive results on Spider benchmark, surpassing GPT-4-based methods and most open-source models, which underscores its strong generalization ability. Nevertheless, the performance gain is relatively modest, as a large portion of SQL queries in the Spider dataset produce empty execution results,

Method	dev(EX)
AskData + GPT-4o	75.36
CHASE-SQL + Gemini	74.46
XiYan-SQL	73.34
OpenSearch-SQL, v2 + GPT-4o	69.30
CHESS	68.31
Distillery + GPT-4o	67.21
MCS-SQL	63.36
MAC-SQL + GPT-4	59.39
DAIL-SQL + GPT-4	54.76
DIN-SQL + GPT-4	50.72
GPT-4	46.35
DTS-SQL + DeepSeek-7B	55.80
SFT CodeS-15B	58.47
SQL-o1 + Llama3-8B	63.40
OneSQL-v0.1-Qwen-32B	64.60
XiYanSQL-QwenCoder-32B	67.01
Qwen2.5-72B-Instruct	60.17
SDE-SQL + Qwen2.5-72B-Instruct	67.67
SDE-SQL (SFT)	68.19

Table 2: The experimental results of competing models on the BIRD dataset.

thereby limiting the effectiveness of feedback from database exploration.

4.3 Efficiency and Latency Analysis

To assess the practical overhead of SDE-SQL, we evaluate end-to-end inference latency under local deployment with vLLM. As shown in Table 4, SDE-SQL achieves an average latency of 35.19s per instance on the full BIRD-dev set, and remains competitive with existing multi-call Text-to-SQL frameworks (34.67s vs. 49.67s for MAG-SQL and 38.25s for CHESS on a sampled subset). We further observe that pre-generation exploration accounts for the largest portion of the runtime, and latency increases with question difficulty, suggesting that harder questions require more extensive database exploration. Overall, these results indicate that

Method	dev(EX)	test(EX)
SDE-SQL (SFT)	87.5	88.5
SDE-SQL + Qwen2.5-72B-Instruct	87.3	88.3
MAC-SQL + GPT-4	86.8	82.8
SENSE-13B	84.1	83.5
SQL-o1 + Llama3-8B	87.4	85.4
DAIL-SQL + GPT-4	84.4	86.6
ROUTE + Qwen2.5-14B	87.3	87.1
DIN-SQL + GPT-4	82.8	85.3
GPT-4 (zero-shot)	73.4	-
Qwen2.5-72B-Instruct	73.9	84.0

Table 3: The experimental results of competing models on the Spider dataset.

Diff.	Size	Tot. (s)	SL (s)	Pre-Exp. (s)	Gen. (s)	Ref. (s)
Simp.	925	33.27	1.36	17.01	8.37	6.53
Mod.	464	37.60	1.30	19.66	9.07	7.57
Chall.	145	39.75	1.20	21.64	9.32	7.59
All	1534	35.19	1.33	18.25	8.67	6.95

Table 4: Latency of SDE-SQL on the BIRD-dev set.

Note: Diff. = difficulty level; Tot. = average total time; SL = schema linking; Pre-Exp. = pre-generation exploration; Gen. = zero-shot generation; Ref. = exploration-based refinement.

SDE-SQL introduces a moderate and practically manageable overhead.

4.4 Ablation Study

For each module in SDE-SQL, we conduct ablation studies on the development set of BIRD benchmark, which is shown in Table 1. In addition, we evaluate the effect of incorporating the fine-tuned explorer and generator into the pipeline. The results demonstrate that each component plays an important role, with the introduction of the two exploration phases leading to particularly significant performance improvements. Besides, modules fine-tuned on their respective sub-tasks can further enhance the overall performance of the workflow.

5 Conclusion

In this work, we propose **SDE-SQL**, a novel Text-to-SQL framework that integrates *Self-Driven Exploration* into both the SQL generation and refinement stages. By enabling LLMs to proactively interact with databases through SQL probes, SDE-SQL bridges the gap between static query generation and dynamic, execution-based reasoning. This exploration mechanism allows LLMs to uncover latent schema semantics and execution patterns,

significantly improving their ability to produce executable and semantically accurate SQL queries.

Extensive experiments on the BIRD and Spider datasets demonstrate the effectiveness of SDE-SQL, with the model achieving an execution accuracy of **68.19%** on BIRD after supervised fine-tuning. Ablation studies confirm the contributions of key components—especially the exploration pipeline and the fine-tuning strategies. As future work, we plan to explore tighter integration of exploration signals into model training to further strengthen the model’s reasoning capabilities.

6 Limitations

Although self-driven exploration significantly enhances the potential of large language models in Text-to-SQL tasks, our current approach has several limitations. In SDE-SQL, database exploration is entirely prompt-driven—meaning that the effectiveness of the exploration process heavily depends on the design and quality of manually crafted prompts. Poorly constructed prompts may lead the model to generate uninformative or redundant SQL probes, thereby limiting its ability to acquire meaningful schema knowledge or execution insights. Moreover, relying solely on prompt engineering can restrict the model’s capacity for deeper reasoning, as it lacks mechanisms for adaptive learning based on feedback from the environment.

Another limitation is the model’s inability to autonomously refine its exploration strategy over time. Since each SQL probe is generated statically from prompts, the model cannot dynamically adjust its behavior based on prior successes or failures during the exploration process. This constraint reduces the overall flexibility and learning efficiency of the system.

To address these issues, future work will focus on making database exploration more intrinsic to the model itself. One promising direction is to incorporate reinforcement learning or other feedback-driven learning paradigms, allowing the model to iteratively refine its probing strategies based on execution outcomes. By enabling the model to learn from its own interactions with the database, we hope to develop a more robust, adaptive framework capable of deeper, context-aware reasoning in complex database environments.

7 Ethical Considerations

Text-to-SQL systems may raise privacy and security concerns when deployed on real-world databases, since natural-language queries and generated SQL may expose sensitive database content or access unintended fields. In our experimental setting, all models are locally deployed open-source models rather than third-party hosted APIs, which reduces the risk of sending database content to external services. However, this does not eliminate all deployment risks; practical use still requires proper access control, sandboxed execution, and human oversight. This is also an important issue that future research in this area should continue to examine.

Acknowledgements

Part of this work was completed during an internship at the Guangdong Laboratory of AI and Digital Economy (SZ), Shenzhen, China. The authors gratefully acknowledge the support and research environment provided during this internship. ChatGPT was used solely for language polishing during manuscript preparation, including minor paraphrasing and wording refinement. All technical content, claims, and final wording were verified by the authors.

References

- Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. [Sadga: Structure-aware dual graph aggregation network for text-to-sql](#). In *Advances in Neural Information Processing Systems*, volume 34, pages 7664–7676. Curran Associates, Inc.
- Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. [LGE SQL: 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, Online. Association for Computational Linguistics.
- Xiaojun Chen, Tianle Wang, Tianhao Qiu, Jianbin Qin, and Min Yang. 2024. [Open-sql framework: Enhancing text-to-sql on open-source large language models](#). *arXiv preprint arXiv:2405.06674*.
- DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2021. [RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases](#). *Computational Linguistics*, 47(2):309–332.
- Yaxun Dai, Wenxuan Xie, Xialie Zhuang, Tianyu Yang, Yiying Yang, Haiqin Yang, Yuhang Zhao, Pingfu Chao, and Wenhao Jiang. 2025a. [Reex-sql: Reasoning with execution-aware reinforcement learning for text-to-sql](#). *CoRR*, abs/2505.12768.
- Yaxun Dai, Haiqin Yang, Mou Hao, and Pingfu Chao. 2025b. [PARSQL: Enhancing text-to-SQL through SQL parsing and reasoning](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 661–681, Vienna, Austria. Association for Computational Linguistics.
- Mingwen Dong, Nischal Ashok Kumar, Yiqun Hu, Anuj Chauhan, Chung-Wei Hang, Shuaichen Chang, Lin Pan, Wuwei Lan, Henghui Zhu, Jiarong Jiang, Patrick Ng, and Zhiguo Wang. 2025. [PRACTIQ: A practical conversational text-to-SQL dataset with ambiguous and unanswerable queries](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 255–273, Albuquerque, New Mexico. Association for Computational Linguistics.
- Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. [Text-to-sql empowered by large language models: A benchmark evaluation](#). *Proc. VLDB Endow.*, 17(5):1132–1145.
- Pushpendu Ghosh, Aryan Jain, and Promod Yenigalla. 2025. [SQLGenie: A practical LLM based system for reliable and efficient SQL generation](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 1004–1012, Vienna, Austria. Association for Computational Linguistics.
- Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jianguang Lou, Ting Liu, and Dongmei Zhang. 2019. [Towards complex text-to-SQL in cross-domain database with intermediate representation](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.
- Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. 2024. [Knowledge-to-SQL: Enhancing SQL generation with data expert LLM](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10997–11008, Bangkok, Thailand. Association for Computational Linguistics.
- Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, and Xiaodan Zhu. 2021. [Improving text-to-sql with schema dependency learning](#). *arXiv preprint arXiv:2103.04399*.
- Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. [A comprehensive exploration on wikisql with table-aware word contextualization](#). *arXiv preprint arXiv:1902.01069*.

- Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. [MCS-SQL: Leveraging multiple prompts and multiple-choice selection for text-to-SQL generation](#). In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353, Abu Dhabi, UAE. Association for Computational Linguistics.
- Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. [Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows](#). In *International Conference on Learning Representations*, volume 2025, pages 28691–28735.
- Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. [Codes: Towards building open-source language models for text-to-sql](#). *Proc. ACM Manag. Data*, 2(3).
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi 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. 2023. 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*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.
- Xiping Liu and Zhao Tan. 2023. Divide and prompt: Chain of thought prompting for text-to-sql. *arXiv preprint arXiv:2304.11556*.
- Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. 2026. [Xiyang-sql: A novel multi-generator framework for text-to-sql](#). *IEEE Transactions on Knowledge and Data Engineering*, 38(4):2474–2487.
- OpenAI. 2024. [Gpt-4o system card](#). *CoRR*, abs/2410.21276.
- Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Arik. 2025. [Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql](#). In *International Conference on Learning Representations*, volume 2025, pages 60385–60415.
- Mohammadreza Pourreza and Davood Rafiei. 2023. [Din-sql: Decomposed in-context learning of text-to-sql with self-correction](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 36339–36348. Curran Associates, Inc.
- Mohammadreza Pourreza and Davood Rafiei. 2024. [DTS-SQL: Decomposed text-to-SQL with small large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, Miami, Florida, USA. Association for Computational Linguistics.
- Mohammadreza Pourreza, Ruoxi Sun, Hailong Li, Lesly Miculicich, Tomas Pfister, and Sercan O Arik. 2024. [Sql-gen: Bridging the dialect gap for text-to-sql via synthetic data and model merging](#). *arXiv preprint arXiv:2408.12733*.
- Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.
- Jiawei Shen, Chengcheng Wan, Ruoyi Qiao, Jiazhen Zou, Hang Xu, Yuchen Shao, Yueling Zhang, Weikai Miao, and Geguang Pu. 2025. A study of in-context-learning-based text-to-sql errors. *arXiv preprint arXiv:2501.09310*.
- Ruoxi Sun, Sercan O Arik, Alexandre Muzio, Lesly Miculicich, Satya Kesav Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. 2024. [SQL-paLM: Improved large language model adaptation for text-to-SQL](#). *Transactions on Machine Learning Research*.
- Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. [Exploring chain of thought style prompting for text-to-SQL](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5376–5393, Singapore. Association for Computational Linguistics.
- Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. [Chess: Contextual harnessing for efficient sql synthesis](#). *arXiv preprint arXiv:2405.16755*.
- Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. [RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.
- Bing Wang, Yan Gao, Zhoujun Li, and Jian-Guang Lou. 2023. [Know what I don't know: Handling ambiguous and unknown questions for text-to-SQL](#). In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5701–5714, Toronto, Canada. Association for Computational Linguistics.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Ji-qi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. [MAC-SQL: A multi-agent collaborative framework for text-to-SQL](#). In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557, Abu Dhabi, UAE. Association for Computational Linguistics.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le,

- and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.
- Wenxuan Xie, Gaochen Wu, and Bowen Zhou. 2024. Mag-sql: Multi-agent generative approach with soft schema linking and iterative sub-sql refinement for text-to-sql. *arXiv preprint arXiv:2408.07930*.
- Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sql-net: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. [Qwen2.5 technical report](#). *CoRR*, abs/2412.15115.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. [Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.
- John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI’96, pages 1050–1055. AAAI Press.
- Hanchong Zhang, Ruisheng Cao, Hongshen Xu, Lu Chen, and Kai Yu. 2024a. [CoE-SQL: In-context learning for multi-turn text-to-SQL with chain-of-editions](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6487–6508, Mexico City, Mexico. Association for Computational Linguistics.
- Tingkai Zhang, Chaoyu Chen, Cong Liao, Jun Wang, Xudong Zhao, Hang Yu, Jianchao Wang, Jianguo Li, and Wenhui Shi. 2024b. [Sqlfuse: Enhancing text-to-sql performance through comprehensive llm synergy](#). *arXiv preprint arXiv:2407.14568*.

A Examples of Error Cause

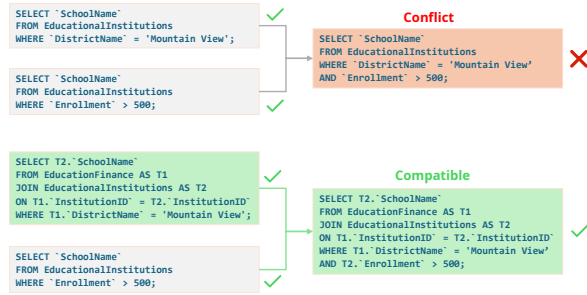


Figure 5: An Example of Condition Conflict

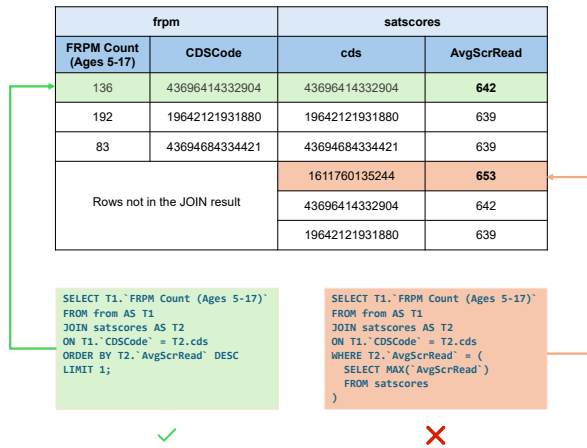


Figure 6: An example of Sub-query Scope Inconsistency

B Target Checking Module

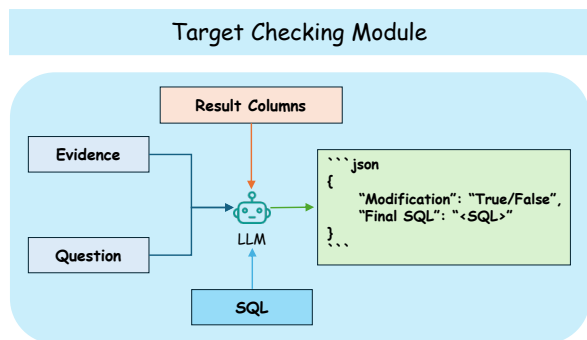


Figure 7: Target Checking Module

C Training Settings

Parameter	Value
per_device_train_batch_size	1
gradient_accumulation_steps	8
learning_rate	1.0e-4
num_train_epochs	2.0
lr_scheduler_type	cosine
lora_rank	16

Table 5: Training hyperparameter configurations.

D Figures

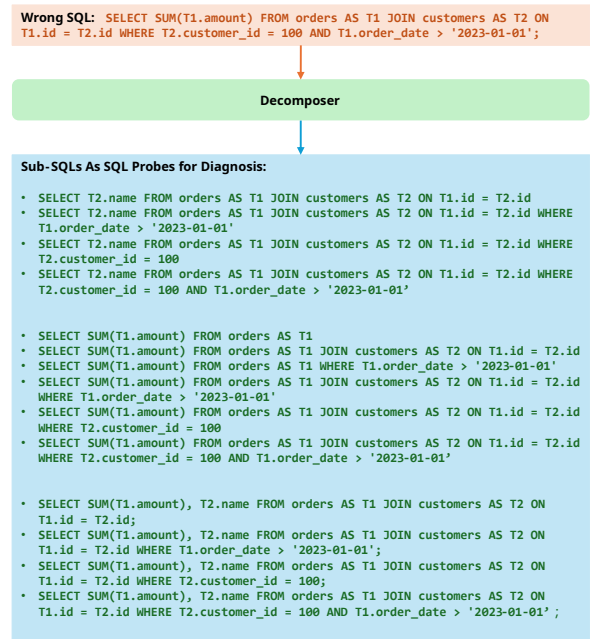


Figure 8: SQL Probes in Error Cause Identification Stage

E Prompt used by SDE-SQL in the training-free setting

For approaches that do not rely on supervised training, it becomes particularly crucial to carefully craft and design prompt templates that can effectively guide the model to carry out Self-Driven Exploration behaviors in a controlled and meaningful manner. In this section, we present a comprehensive set of prompt templates that are utilized across different stages of the SDE-SQL pipeline to support this capability. Due to limitations in available space, certain detailed elements and specific prompt examples have been omitted, but the essential structures and core ideas are fully retained.

[Instruction]

Your task is to generate a series of SQL Probes to explore the database and identify the correct columns mentioned the given question. These Probes will help determine which columns contain the necessary data and ensure that the final SQL query returns non-empty results. Follow these requirements:

[Requirements]

- In this task, you should identify and list all entities mentioned in the question, along with their corresponding candidate columns in the database schema. For each entity, there is only one candidate column unless the database schema contains multiple columns with the same or extremely similar meanings that are consistent with the entity. Do not include unnecessary columns as candidate columns. For each entity, if it corresponds to multiple candidate columns, generate SQL Probes to check the presence of relevant values in each candidate column. If a specific value is mentioned for an entity (e.g., 'Mountain View' district or enrollment > 500), include SQL Probes to verify the existence of that value in the candidate columns.

- The entities in the question are divided into two types: target entity and condition entity. The target entity is the ultimate goal of the query, while the condition entity corresponds to the conditions that the target entity needs to satisfy. First, you need to generate the corresponding Base SQL Probes based on the target entity. Then, for each condition entity, generate the corresponding Condition SQL Probes based on the Base SQL Probes.

Base SQL Probes: At first generate the base SQL Probes that search for the target entity. All other SQL Probes should be generated based on this base SQL Probe.

Condition SQL Probes: Generate SQL Probes for each condition entity based on the Base SQL Probe.

[Attention]

- If the [Evidence] specifies a candidate column or candidate value for an entity, use that column or value as the mapping for the entity directly if [Evidence] is reasonable, and there is no need to explore other candidates. If there is a calculation formula for an entity in the [Evidence], prioritize using this formula to represent the entity. This is very important!!!

- You don't need to consider SQL Probes that combine multiple conditions.

- Base SQL Probes should only select the targets directly without other conditions.

- Condition SQL Probes will add new conditions to the Base SQL Probe.

[Note]

...

[SQL Tricks]

...

[Database admin instructions]

...

[Output Format]

...

Figure 9: Prompt Template of Candidates Exploration

[Instruction]

The question provided to you can be broken down into a target and several conditions. Previously, a series of SQL Probes based on the target and conditions were generated. Among these, the Base SQL Probes are generated for the target, while the other SQL Probes are based on the Base SQL Probes with the addition of exploring a specific condition. I will provide you with these SQL Probes and their corresponding execution results (whether they return empty or not). What you need to do is combine the conditions based on the Database schema and the question to generate a new series of SQL Probes. This will help conduct a more in-depth exploration of the database and assist me in generating the final SQL for the question.

[Requirements]

- The execution results can be one of two outcomes: NULL or Not NULL. !!!NULL means that the result of the SQL query is empty (no data matches the conditions). Not NULL means that the result of the SQL query is not empty (there is data that matches the conditions)!!!
- You need to analyze the current execution results, eliminate the obviously invalid candidate columns, and only combine the ones that are potentially valid.
- You need to combine all the conditions to ensure a comprehensive exploration. For example, suppose the current question contains a target and three conditions. After analyzing the execution results, the candidate columns are as follows: the unique candidate column for the target can be determined from the Base SQL Probes, the first condition has two possible candidate columns, the second condition has one possible candidate column, and the third condition has three possible candidate columns. Therefore, the number of SQL Probes to be generated after combining them would be $1 * 2 * 1 * 3 = 6$.

[Tips]

...

[Output Format]

...

Figure 10: Prompt Template of Combinations Exploration

```
# Task Description
You are an SQLite database expert tasked with generating a SQL query according
to a input user question. You will be provided:
- An input user question, and potentially an evidence
- The database schema
- The descriptions of columns(column name, data_format, description)
- The value retrieved from database
- The SQL Probe result

Your task is to generate the correct SQL query. The input question consists of
a query target and the conditions that the target needs to satisfy. You need
to analyze the semantics of the question and convert it into the corresponding
SQL. You should imitate human, and solve this task step by step.

# Note
...

# SQL Tricks
...

# Database admin instructions
...

# Output Format
...
```

Figure 11: Prompt Template of Zero-shot Generation

[Instruction]

When executing an SQL statement, there may be instances where the execution result is completely empty.

You need to identify the cause of the error based on the query and database information and generate some new probe SQLs to find solution.

To help you to find out the reason, I extracted a batch of probe SQLs from this incorrect SQL and executed them, providing you with the execution results (NULL or Not NULL). !!!NULL means that the result of the SQL query is empty (no data matches the conditions). Not NULL means that the result of the SQL query is not empty (there is data that matches the conditions)!!!

You can use these Probe SQL query results to determine where the issue lies based on whether they are empty or not.

The revised SQL must be consistent with the Query, and it should not omit any necessary conditions described in the Query.

[Possible Causes]

Cause 1: Conflicting Conditions or Redundant Descriptions Across Different Columns

--details: Conflicting: The simultaneous existence of two conditions leads to null, proving that the column for one of the conditions was chosen incorrectly. Redundant: For a certain condition, multiple different columns are used to repeat the description, resulting in a conflict between this condition and other conditions.

--fix: For the conflicting case, certainly, it seems that a condition might be described by several columns with similar meanings, but the incorrect column was selected in the SQL. To resolve this, identify and replace the column name accordingly. For the Redundant case, remove duplicate descriptions of the same condition and keep the one that fits best.

Cause 2: Incorrect Condition Values or Case Sensitivity Issues

--details: The conditions in the query may use incorrect values or fail to account for case sensitivity when comparing strings.

--fix: Try to use `LIKE` because the LIKE keyword is case-insensitive by default.(table.<column> = 'xxx' -> table.<column> LIKE 'xxx')

Cause 3: Unnecessary Table Joins Resulting in No Satisfying Records.

--details: The query may include unnecessary table joins, resulting in no records satisfying the conditions in the final intersection.

--fix: Check if every table join is really necessary and discard unnecessary tables.

Cause 4: Incorrect Column Selection

--details: Among the several tables involved, there may be multiple candidate columns for a certain condition, but Old SQL selected the wrong one.

--fix: Determine if there is a more suitable column, or use a similar column from another table.

Cause 5: Misuse of the MAX(MIN) function or `ORDER BY`

--details: Using the MAX(MIN) function or `ORDER BY` in a subquery(nested sql), the data corresponding to this maximum or minimum value may not be in the intersection of the two tables, so it may return a null value.

--fix: First JOIN the tables, and then use MAX(MIN) function or `ORDER BY` on the JOIN results.

[Requirements]

After thinking step by step, you may already have some guesses and potential solutions about the cause of the error, but you need to validate these guesses and solutions. Please generate a set of probe SQLs based on your analysis to help your future self arrive at the correct SQL.

Figure 12: Prompt Template of Solution Exploration

[Instruction]

When executing an SQL statement, there may be instances where the execution result is completely empty.

You need to identify the cause of the error based on the query and database information and make the necessary corrections.

To help you to find out the reason, I executed a series of SQLs which is related to this question, providing you with the execution results (NULL or Not NULL). !!!NULL means that the result of the SQL query is empty (no data matches the conditions). Not NULL means that the result of the SQL query is not empty (there is data that matches the conditions)!!!

You can use these Probe SQL query results to determine where the issue lies based on whether they are empty or not.

Note that your modified SQL still has to correspond one-to-one with the targets and conditions in the Query.

[Possible Causes]

Cause 1: Conflicting Conditions or Redundant Descriptions Across Different Columns

-- details: Conflicting: The simultaneous existence of two conditions leads to null, proving that the column for one of the conditions was chosen incorrectly. Redundant: For a certain condition, multiple different columns are used to repeat the description, resulting in a conflict between this condition and other conditions.

-- fix: For the conflicting case, certainly, it seems that a condition might be described by several columns with similar meanings, but the incorrect column was selected in the SQL. To resolve this, identify and replace the column name accordingly.

For the Redundant case, remove duplicate descriptions of the same condition and keep the one that fits best.

Cause 2: Incorrect Condition Values or Case Sensitivity Issues

-- details: The conditions in the query may use incorrect values or fail to account for case sensitivity when comparing strings.

-- fix: Try to use `LIKE` because the LIKE keyword is case-insensitive by default.(table.<column> = 'xxx' -> table.<column> LIKE 'xxx')

Cause 3: Unnecessary Table Joins Resulting in No Satisfying Records.

-- details: The query may include unnecessary table joins, resulting in no records satisfying the conditions in the final intersection.

-- fix: Check if every table join is really necessary and discard unnecessary tables.

Cause 4: Incorrect Column Selection, No Matching Values

-- details: The query may select the wrong column or the column may not have any values that satisfy the condition.

-- fix: Determine if there is a more suitable column, or use a similar column from another table.

Cause 5: Misuse of the MAX(MIN) function or `ORDER BY`

-- details: Using the MAX(MIN) function or `ORDER BY` in a subquery(nested sql), the data corresponding to this maximum or minimum value may not be in the intersection of the two tables, so it may return a null value.

-- fix: First JOIN the tables, and then use MAX(MIN) function ORDER BY on the JOIN result, not in a subquery(nested query).

Figure 13: Prompt Template of Final Refinement

[Instruction]

You are a helpful assistant. Given a question, a SQL statement and probably a corresponding evidence, you need to determine whether the query goal of this SQL and the question are consistent.

[Requirement]

1. First, you need to identify the actual entity (target column) that the question is trying to query.
2. Determine how many columns the question expects to see in the result.
3. Compare the number of columns returned by the SQL query with the expected number of columns to determine if extra columns were selected.
4. If extra columns were selected, you need to modify the target after the SELECT keyword in the original SQL statement to remove the unnecessary target column. If you believe the selected columns are correct or insufficient, no modification is needed.
5. Your output should be in JSON format:
```json  
{  
 "Modification": "<True or False>",  
 "Final SQL": "<sql>"  
}```

[Example]

...

[Attention]

Only modify when you are absolutely certain that there are extra target columns. If you feel there is no issue or are unsure whether there is an issue, do not make any changes.

Figure 14: Prompt Template of Target Checking