Enhancing NLIDBs: Advancing from Text-to-SQL to Text-to-Multi-SQL

Anonymous ACL submission

Abstract

Text-to-SQL is a key part of Natural Language Interfaces to Databases (NLIDB), helping non-technical users query databases. However, it has one significant limitation: it assumes users know that SQL queries return data in a table format. This often leads to problems when users ask for data that cannot be handled by a single query. To address this, we propose Text-to-Multi-SQL, which uses multiple SQL queries to meet complex user needs. We created Spider𝕊, the first Text-to-Multi-SQL dataset, based on the Spider dataset. It includes 7,000 training examples, 1,024 validation examples, and 2,147 test examples, created using a mix of manual and GPT4o generated data. Tests show that current models struggle with multiple SQL generation, and even advanced models perform worse (0.167-0.327 drop in accuracy) on Spider𝕊. We also found that models are very sensitive to prompts—switching from "generate one SQL" to "generate one or multiple SQL" significantly reduces their performance.

1 Introduction

The research on NLIDB (Natural Language Interface to DataBases) has always been a hot topic in the fields of databases and Natural Language Processing (NLP) (Abbas et al., 2022; Pourreza and Rafiei, 2023; Hong et al., 2024; Zhang et al., 2024). In the study of NLIDB, Text-to-SQL is considered one of the key components (Abbas et al., 2022; Qin et al., 2022; Deng et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023a; Hong et al., 2024; Zhang et al., 2024). Many studies suggest that Text-to-SQL technology allows users without database knowledge to easily access data from a database (Iacob et al., 2020; Abbas et al., 2022; Qin et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023a; Gao et al., 2023; Pourreza et al., 2024; Hong et al., 2024; Zhang et al., 2024; Zhu et al., 2024).

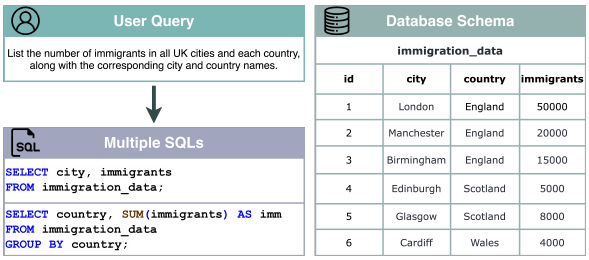

Figure 1: An example of Text-to-Multi-SQL.

However, in our practice, we discovered an implicit assumption in Text-to-SQL: users are expected to understand that the data returned by each SQL query is expected to be a tabular format, typically a rectangular table with a consistent column structure (though there may be variations, such as missing values in sparse data or more complex structures in nested data). For many users without SQL knowledge, they are unaware of this assumption, which leads to the problem that their requests often cannot be fulfilled by a single SQL query, or even by SQL queries at all. This can cause a problem: users may request data that exceeds the capabilities of a single SQL query.

Figure 1 shows a practical example from an NLIDB-based census flow data platform. The data the user needs cannot be fulfilled by a single SQL query; however, two SQL queries can perfectly meet the user's needs. Although it's possible to force the query to be completed with a single SQL, the returned data is neither elegant nor easy to understand. Both we and the users agree that using two SQL queries to return two separate result sets would be a better approach.

Based on this finding, we believe it is necessary to extend traditional Text-to-SQL to Text-to-Multi-SQL to better meet the diverse data needs of users. To this end, we chose the Spider (Yu et al., 2018) Text-to-SQL dataset as the basis and constructed the first Text-to-Multi-SQL dataset: Spider𝕊. Like

the Spider dataset, Spider𝕊 contains 7,000 training examples, 1,024 validation examples, and 2,147 test examples. The data was constructed using a combination of manual annotation and data generated by Large Language Models (LLMs), with around 1,100 manually annotated examples used to guide the LLM in generating additional paired data, where each question could be paired with multiple SQL queries. During generation, we rewrote the original Spider questions and SQLs to ensure the diversity of the Spider𝕊 dataset. To the best of our knowledge, Spider𝕊 is the first Text-to-Multi-SQL dataset, and this paper is the first dedicated to the study of Text-to-Multi-SQL.

Experimental results show that many existing Text-to-SQL models are unable to generate multiple SQL queries. While LLM-based Text-to-SQL models can output multiple SQL queries by modifying the prompt, their performance drops significantly on the Spider𝕊 dataset because they may not be trained on a Text-to-Multi-SQL dataset. The absolute drop in execution accuracy ranges from 0.167 to 0.327. Interestingly, most models show poor robustness to prompts that request one or multiple SQL queries. When we simply changed the prompt from 'generate one SQL' to 'generate one or multiple SQL,' the performance of most models on the original Spider dataset decreased to varying degrees, with the largest drop seen in the CHESS (Talaei et al., 2024) model—an SOTA open source Text-to-SQL model on the BIRD (Li et al., 2024)—whose execution accuracy dropped from 0.759 to 0.462.

Overall, the contributions of this paper are as follows:

1. We systematically study Text-to-Multi-SQL and construct the first Text-to-Multi-SQL dataset, Spider𝕊, based on the Spider Text-to-SQL dataset.

2. We found that Text-to-SQL models are highly vulnerable to prompt modifications: simply changing the instruction from 'generate one SQL' to 'generate one or multiple SQLs' leads to a significant performance drop.

3. We found that existing Text-to-SQL models/LLMs fail to address the need for Text-to-Multi-SQL, with some unable to generate multiple SQL queries. Even capable models show a significant accuracy drop on Spider𝕊 compared to the Spider dataset.

2 Relate Work

2.1 Natural Language Interface to Databases

Research in Natural Language Interface to Databases (NLIDB) focuses on enabling users to query databases using natural language, with key tasks including Text2SQL, Text-to-Graph, and Text-to-SPARQL. The Text2SQL (Katsogiannis-Meimarakis and Koutrika, 2023b) task, which converts natural language questions into SQL queries, is the most prominent, addressing challenges such as query intent interpretation, schema mapping, and handling complex SQL constructs like joins and subqueries. Recent advancements in deep learning, particularly transformer (Vaswani, 2017) and large language (Devlin, 2018; Achiam et al., 2023) models, have significantly improved performance (Scholak et al., 2021; Qi et al., 2022; Li et al., 2023; Gao et al., 2024b) in this area. In addition to Text2SQL, other tasks such as Text-to-Graph (Lawrence, 2024) and Text-to-SPARQL (Luo et al., 2023) aim to extend NLIDB systems to graph and semantic web databases, each introducing unique challenges related to graph structure and ontology-based querying. There is a growing emphasis on improving related models' ability to handle more complex domain-specific queries and to provide accurate context-sensitive responses.

2.2 Popular Text-to-SQL Datasets

As one of the most important studies in human-computer interaction systems, Text2SQL has attracted many people to explore. Several popular databases are proposed for converting natural language queries into structured query language. WikiSQL (Zhong et al., 2017) is a large-scale dataset for converting natural language questions into SQL queries, containing 80,000 question-SQL pairs over tables extracted from Wikipedia. Spider (Yu et al., 2018) is a large-scale, cross-domain dataset for SQL query generation, with a focus on multi-table queries, complex conditions, and aggregation. BIRD (Li et al., 2024) is a benchmark for evaluating multi-lingual and cross-lingual SQL query generation models, particularly focusing on database retrieval across different languages. NL2GQL (Zhou et al., 2024) is a dataset that pairs natural language questions with corresponding GQL queries, designed to help develop models for querying graph databases, including tasks like node selection, graph traversal, and pathfinding.

SQL-eval[1] is an open-source PostgreSQL evaluation dataset constructed based on Spider (Yu et al., 2018), which is used to evaluate Text-to-SQL models' ability to generate executable SQL queries from natural language questions. However, all of the above datasets focus on convert one natural language question into one SQL query. It does not apply to many real-world scenarios where the answer to the user's question should come from multiple SQL.

2.3 Limitations of Text-to-SQL Systems

Recent studies have advanced Text-to-SQL systems from several complementary angles. However, it's still hard for these methods to address complex tasks. Biswal et al. (2024) integrate Retrieval-Augmented Generation (RAG) with database tables to propose a unified, general-purpose paradigm for answering natural language questions over databases, aiming to provide a better user experience than traditional text-to-SQL approaches. Likewise, CHASE (Ma et al., 2025) enhances hybrid query execution by bridging structured and unstructured data. Renggli et al. (2025) analyze the limitations of Text-to-SQL from an evaluation perspective, which can lead to both prediction and evaluation errors. However, none of these papers involved Text-to-Multi-SQL.

3 Task Definition

Traditional Text-to-SQL approaches typically assume a one-to-one mapping between question and SQL query. However, in many real-world applications, a single query is insufficient to fully capture the intent of a complex question. For example, multi-faceted questions often require retrieving information from different perspectives, applying multiple filtering conditions, or aggregating distinct sets of records.

Text-to-Multi-SQL aims to convert a natural language question Q into one or multiple SQL queries $y_1, y_2, \ldots$ that retrieve the correct results from a given database D. A database is formally defined as $D = <C, T>$, where $C = \{c_1, c_2, ..., c_m\}$ represents the set of column names and $T = \{t_1, t_2, ..., t_m\}$ represents the set of table names. Given D and Q, the objective is to generate the most appropriate SQL query (or queries set), such that executing $y_1, y_2, \ldots$ on D yields the expected answer. The query generation process can be expressed as

$$\{y_1, y_2, \ldots\} = f(Q, D|\theta), \quad (1)$$

where $f(\cdot|\theta)$ is a model parameterized by θ, which maps the natural language question Q and database schema D to a set of SQL queries $\{y_1, y_2, \ldots\}$.

For some questions that can be solved using either multiple SQL queries or a single SQL query, we prepare two sets of gold SQL queries. As long as the prediction matches any one of the gold SQL sets, we consider the task successful. For example, in all cases generated by the 6th generation strategy in Appendix A, we have prepared two gold SQL sets for evaluation. However, during training, we use the multi-SQL gold SQL set because its query results better align with the user's intent.

4 Dataset Construction

Figure 2 illustrates our data annotation process.

4.1 Question and SQL Annotation

SQL pattern coverage. Before beginning the annotation process, our team surveyed multiple SQL queries commonly found in real-world applications and identified 14 distinct patterns (e.g., requiring an aggregated result and columns before aggregated (type 6), grouping the same table based on different attributes (type 7), and sorting the same data based on different ordering (type 9) in Appendix A). Each pattern was carefully documented in our annotation manual, detailing the associated SQL clauses and typical ways users might phrase corresponding questions. This approach ensured that the final dataset would capture a broad spectrum of query complexities and structures.

Question clarity. Consistent with the Spider dataset, we aimed to avoid ambiguous questions that either lacked sufficient details or required knowledge outside the database to answer. Instead, we focused on queries that explicitly stated the necessary conditions and returned values. For example, the question "Find all the teacher names and also all the classrooms for math courses" can be interpreted in two ways: either as a request for teachers and classrooms for math courses, or as separate requests for all teacher names and all math course classrooms. To avoid ambiguity, we use phrases like "first ..., then ..." or "separately list ..." to clarify that multiple distinct SQL queries are needed.

[1] https://github.com/defog-ai/sql-eval

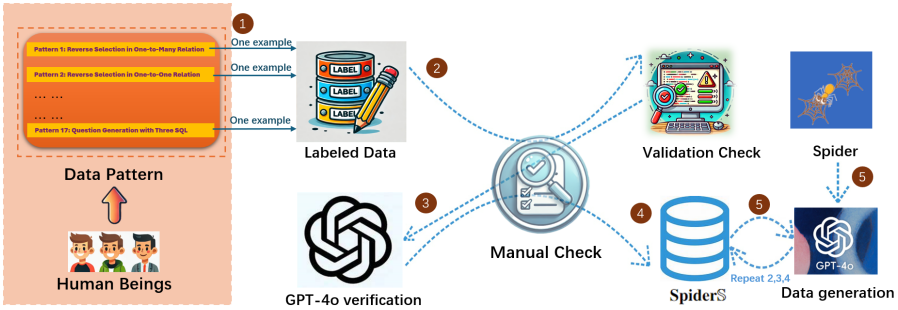

Figure 2: Data annotation process.

Annotation steps. We began by creating a detailed data annotation manual that described each of the 14 SQL patterns, including examples of both questions and their corresponding SQL statements. Two Computer science students(paper authors) studying in the US, all proficient in SQL, were then tasked with labeling one example for each pattern across different databases. These examples served as references for One-Shot Learning in later data generation stages.

After the initial round of labeling, we conducted a manual cross-check to detect obvious inconsistencies or errors. Next, the annotated data was run through an automatic validation script (described in Section 4.2) that checked structural correctness (e.g., matching pattern design, proper use of SQL keywords) and basic logical consistency. Finally, we used GPT-4o to assess whether each user question was clearly stated and matched the corresponding SQL queries accurately. Any data flagged during this process was subjected to a second manual review to resolve potential issues.

Annotation tools. In order to conduct data annotation more efficiently, we establish a website which display the database schema and provide some possibly useful examples as annotation references. After modifying or adding SQLs on the webpage, we can directly execute the SQLs and check whether the result is as expected. The website also is connected to GPT-4o, which may provide meaningful suggestions to make the modified questions without ambiguity. The website also provides data comparison, allowing users to intuitively view the differences between the original questions and the new questions, facilitating modification and review.

4.2 Data Review

Automatic validation script. To maintain annotation consistency and reduce human error, we developed and iteratively refined an automatic validation script. The script performed several checks:

- **Syntactic correctness:** Verified that all SQL queries could be parsed and executed without errors.
- **Pattern matching:** Ensured each query corresponded to one of the 15 predefined SQL patterns and did not include extraneous or missing clauses.
- **Schema alignment:** Checked that table names and column names used in the SQL were present in the database schema.

Entries that did not pass any of these checks were returned to the annotators for correction or deletion.

GPT4o-Based validation. In addition to the rule-based checks, we employed GPT-4o as a second layer of validation. GPT-4o was given a pair consisting of a user query and its corresponding SQL statement. It was then asked to:

1. Evaluate whether the question was sufficiently clear and unambiguous.
2. Verify if the SQL query logically aligned with the question.
3. Flag any potential misalignment or unclear phrasing.

Data flagged by GPT-4o for potential issues—such as vague question wording or mismatches in conditions—was forwarded to a human reviewer for final judgment. This hybrid workflow ensured a more robust validation process than either method could achieve in isolation.

4.3 Construct the Final Dataset

After one round of manual validation, one round of automatic validation, and one round of GPT-4o

SQL keyword	Spider	Spider𝕊
WHERE	51.37%	52.34%
GROUP	29.02%	31.66%
ORDER	15.70%	14.93%
LIMIT	39.40%	35.10%
UNION	1.02%	0.11%
INTERSECT	3.91%	0.22%
EXCEPT	3.49%	0.37%
JOIN	58.55%	39.33%

Table 1: Percentage of SQL keywords in Spider and Spider𝕊.

verification, we obtained a high-quality set of annotated question-SQL pairs. Then, we generated additional data using GPT-4o based on One-Shot learning if a proper example was found; otherwise, we used the Zero-Shot learning approach to generate extra data.

We expanded the dataset to about twice the size of the original Spider dataset and applied the same multi-step validation process, including automatic checks, manual review, and GPT-4o-based validation, to ensure accuracy and consistency. For most data that failed automatic validation or GPT-4o checks, we removed it directly to save annotation time. Through manual inspection and review, we ensured that the new dataset remained consistent in quality with the original Spider dataset. To maintain balance, we set a limit so that no single SQL pattern made up more than 15% of the dataset. This prevented overrepresentation of certain query types and ensured diversity. After these rigorous validation steps, we produced a well-verified, high-quality Text-to-Multiple-SQL dataset.

4.4 Dataset Statistics

Our dataset follows the same structure as the Spider dataset, with 7,000 training samples, 1,034 validation samples, and 2,147 test samples. Of these, 1,130 examples were manually created, and the remaining 9,051 were automatically generated. On average, each question requires 2.125 SQL queries to answer. Specifically, 1,266 samples need 3 SQL queries, while 8,915 samples require 2. Table 1 compares the frequency of SQL keywords between the two datasets. Compared to the original Spider, Spider𝕊 has similar frequencies for keywords like WHERE, GROUP, ORDER, and LIMIT. However, JOIN, UNION, INTERSECT, and EXCEPT are less frequent in Spider𝕊 because we opted for simpler SQL queries in Spider𝕊, aiming to avoid overly complex queries, as multiple complex queries are less common in real user scenarios.

5 Evaluation Metrics

Execution Score (ES). In order to evaluate the correctness of each of the multiple generated SQL queries, we propose a metric called Execution Score (ES). For the i^{th} evaluation question requiring multiple SQL queries to address, given P_i and G_i as the executed result sets of the predicted SQLs and the ground-truth SQLs separately, ES_i is defined as the intersection of the result set P_i and G_i, relative to the maximum between the size of the two result sets. This metric is applied to evaluate the ability of generating meaningful and correct SQL queries matching part of the question. Therefore, we consider the correctness for each single predicted SQL and assign a higher score if the executed results of predicted SQLs match more ground-truth ones. The final ES is the accumulation of ES_i, which is shown below:

$$ES_i = \frac{|P_i \cap G_i|}{\max(|P_i|, |G_i|)}, \tag{2}$$

where:

- P_i : The set of the i^{th} predicted SQL queries.
- G_i : The set of the i^{th} ground-truth SQL queries.
- $P_i \cap G_i$: The set of SQL query pairs (p, g) for the i^{th} question such that the execution result of p and g is identical.
- $|\cdot|$: The cardinality (size) of a set.

Execution Accuracy (EX). EX is formally defined as the proportion of examples in the evaluation set for which the executed results of both the predicted and ground-truth SQLs for a single question are identical, relative to the overall number of SQLs. However, in our task definition, we require multiple SQLs to address one single question. Therefore, following the previous definition of Execution Score (ES), the i^{th} question in the evaluation set is considered solved only when ES_i is equal to 1. That is, the execution result of each of the predicted SQLs can match the one of a corresponding ground-truth SQL. The computation of EX can be expressed as follows:

$$\text{EX} = \frac{\sum_{i=1}^{N} \mathbb{M}(ES_i)}{N}, \tag{3}$$

Models	Spider		Spider$\mathbb{M}$		Spider$\mathbb{S}$		Spider$_{small}$		Spider$\mathbb{M}_{small}$		Spider$\mathbb{S}_{small}$	
	EX	ES	EX	ES	EX	ES	EX	ES	EX	ES	EX	ES
Closed-source Models												
GPT-4o	-	-	-	-	0.497	0.648	0.733	0.733	0.572	0.625	0.519	0.667
Claude-3.5-Sonnet	-	-	-	-	-	-	0.629	0.629	0.589	0.589	0.302	0.461
o3-mini	-	-	-	-	-	-	0.737	0.737	0.722	0.723	0.525	0.650
Open-source Models												
Qwen2.5-Coder-7B-Instruct	0.756	0.756	0.674	0.694	0.442	0.596	0.761	0.761	0.678	0.697	0.454	0.608
Qwen2.5-Coder-32B-Instruct	0.766	0.766	0.675	0.695	0.503	0.647	0.779	0.779	0.689	0.705	0.515	0.659
CodeLlama-7b-Instruct-hf	0.123	0.123	0.082	0.089	0.038	0.069	0.125	0.125	0.070	0.078	0.033	0.065
CodeLlama-34b-Instruct-hf	0.177	0.177	0.221	0.224	0.088	0.141	0.172	0.172	0.216	0.218	0.086	0.140
Llama-3.1-8B-Instruct	0.628	0.628	0.621	0.628	0.392	0.549	0.622	0.622	0.615	0.620	0.411	0.567
Llama-3.1-405B-Instruct	-	-	-	-	-	-	0.787	0.787	0.748	0.750	0.534	0.688
DeepSeek-R1-Distill-Qwen-7B	0.394	0.394	0.320	0.347	0.195	0.322	0.375	0.375	0.300	0.330	0.195	0.330
DeepSeek-R1-Distill-Llama-8B	0.435	0.435	0.388	0.404	0.125	0.283	0.415	0.415	0.374	0.387	0.137	0.293
Mistral-Large-Instruct-2411	-	-	-	-	-	-	0.671	0.671	0.454	0.522	0.453	0.634
Fine-tuned Models												
Qwen2.5-Coder-7B-Instruct	-	-	0.786	0.787	0.725	0.831	-	-	0.784	0.785	0.731	0.837
CodeLlama-7b-Instruct-hf	-	-	0.728	0.728	0.656	0.781	-	-	0.728	0.729	0.651	0.781
Llama-3.1-8B-Instruct	-	-	0.761	0.761	0.706	0.811	-	-	0.753	0.753	0.724	0.822
DeepSeek-R1-Distill-Qwen-7B	-	-	0.664	0.664	0.604	0.734	-	-	0.660	0.661	0.615	0.744
DeepSeek-R1-Distill-Llama-8B	-	-	0.728	0.729	0.665	0.788	-	-	0.730	0.730	0.672	0.793
Text2SQL Models												
CHESS	-	-	-	-	-	-	0.759	0.759	0.462	0.589	0.559	0.699
XiYanSQL-QwenCoder-32B	0.849	0.849	0.809	0.814	0.669	0.782	0.852	0.852	0.823	0.827	0.685	0.799

Table 2: Performance Comparison of Various Models on Spider and Spider$\mathbb{S}$ Datasets.

where function $\mathbb{M}(\cdot)$ returns 1 only if ES_i is equal to 1, indicating that every executed result in predicted result set P_i can match a corresponding one in ground-truth result set G_i. The expression of $\mathbb{M}(\cdot)$ is shown below:

$$\mathbb{M}(ES) = \begin{cases} 1, & \text{if } ES_i = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

6 Experiments

6.1 Experimental Setup

Prompt. In terms of prompt design, the distinction lies in generating one SQL versus one or more SQL. When evaluating Spider$\mathbb{S}$, the latter must be used, whereas for Spider, both approaches are applicable. To analyze the impact of prompts, we employ different prompts on Spider to observe how the model's performance is affected when the generated SQL is no longer restricted to a single query.

Dataset. We conducted our experiments in both the Spider and Spider$\mathbb{S}$ test sets. The test set distribution of the original Spider dataset consists of 470 easy, 857 medium, 463 hard, and 357 extra-hard samples, totaling 2,147 queries. In our Spider$\mathbb{S}$ dataset, the distribution has slightly shifted, with 351 easy, 837 medium, 595 hard, and 364 extra-hard samples, while still maintaining a total of 2,147 queries. Despite these minor variations—such as a decrease in easy queries (470 $\rightarrow$ 351) and an increase in hard queries (463 $\rightarrow$ 595)—the overall difficulty distribution remains similar across both datasets.

To improve evaluation efficiency and reduce evaluation costs, we randomly sampled 1,000 examples each from the original Spider and Spider-S test sets to create a small test set for low-cost evaluation.

The evaluation tasks used in this study are shown as follows:

- **Spider$\mathbb{S}$**: The complete Spider$\mathbb{S}$ test set(2,147-item).
- **Spider$\mathbb{S}_{small}$**:The small Spider$\mathbb{S}$ test set(1,000-item).
- **Spider**: The complete Spider test set(2,147-item) with a prompt for generating one SQL.
- **Spider$_{small}$**: The small Spider test set (1,000-item) with a prompt for generating one SQL.
- **Spider$\mathbb{M}$**: The complete Spider test set(2,147-item) with a prompt for generating one or multi

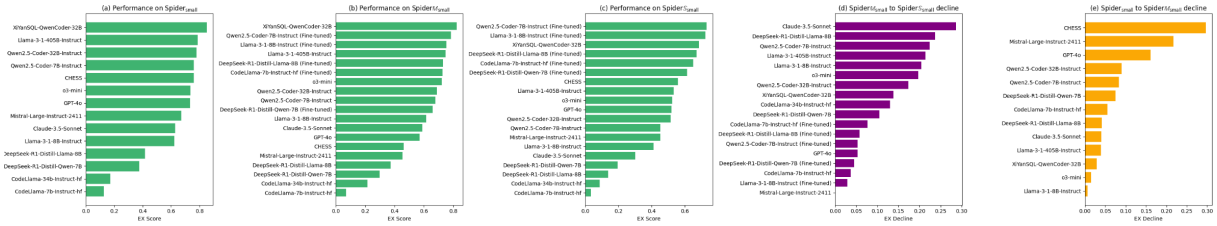

Figure 3: Model EX rankings on the Spider(a), Spider$\mathbb{M}$(b), and Spider$\mathbb{S}$(c), and the EX drop caused by task migration(d,e).

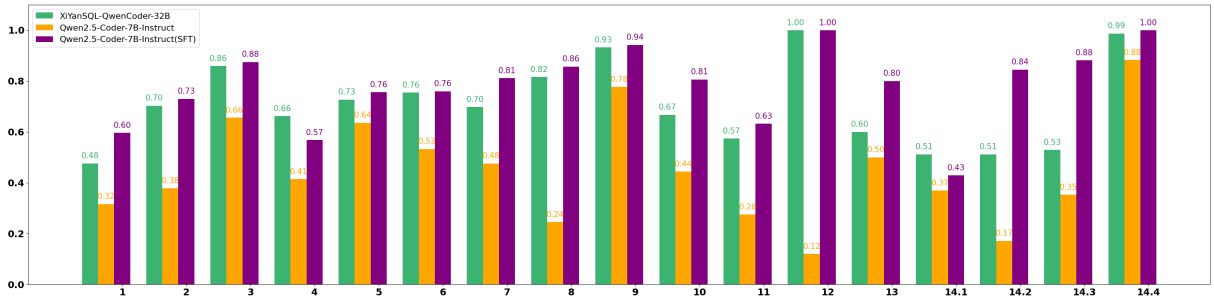

Figure 4: EX scores for different question types.

SQL.

- **Spider**$\mathbb{M}_{small}$:The small Spider test set (1,000-item) with a prompt for generating one or multi SQL.

Models. To comprehensively evaluate the Text-to-Multi-SQL capabilities of different models, we assessed a diverse set of models, including open-source, fine-tuned, closed-source, and specialized Text2SQL models. For open source models, we study LLama3.1 (Grattafiori et al., 2024), Qwen 2.5 Coder (Qwen et al., 2025), DeepSeek-R1-Distill-models(DeepSeek-AI et al., 2025), CodeLlama (Rozière et al., 2024). Among the closed-source model, we evaluate GPT-4o (OpenAI et al., 2024), Claude-3.5-Sonnet, and o3-mini. For text-to-SQL models, we choose CHESS (Talaei et al., 2024) and XiYanSQL-QwenCoder-32B (Gao et al., 2024a).

6.2 Main Results

Table 2 compares the performance of different models on the Spider and Spider$\mathbb{S}$ datasets. Overall, XiYanSQL-QwenCoder-32B, which is specifically optimized for SQL generation, performs the best. Among the other models, Llama-3.1-405B-Instruct achieves the best results, with o3-mini close behind. The model with the lowest performance is CodeLlama-34B-Instruct, mainly due to its inability to generate correct JSON output and the significant hallucinations it produces.

While the highest EX score on the Spider dataset is 0.85, even models fine-tuned specifically for Spider$\mathbb{S}$ only reach a maximum EX score of 0.731. This suggests that there is still room for improvement on Spider$\mathbb{S}$. More importantly, our experiments show that many models struggle with robustness when generating multiple SQL queries, which we discuss in more detail in Section 6.3.

Comparison of Small and Full Datasets. When comparing the full test set to its subset, we find that the results are very similar, with the small dataset slightly outperforming the larger one. This shows that our results on the small dataset are reliable.

Model Size and Performance. There is a clear positive correlation between model size and performance. For example, in the Llama-3.1 series, the 8B-parameter achieves an EX score of 0.411 on the **Spider**$\mathbb{S}_{small}$, while the 405B-parameter model achieves a score of 0.534. A similar trend is seen in the Qwen2.5 series, where the 32B model outperforms the 7B model by 0.059 in EX score on the **Spider**$\mathbb{S}_{small}$.

Effectiveness of Fine-Tuning. Comparing model performance before and after fine-tuning, we find that domain-specific fine-tuning leads to significant improvements. For example, CodeLlama-7B's EX score on the **Spider**$\mathbb{S}$ increased from 0.082 to 0.728, a 788% improvement.

6.3 Robustness Analysis

Almost all models show a drop in performance when moving from Spider to Spider$\mathbb{M}$ or from Spider$\mathbb{M}$ to Spider$\mathbb{S}$. The performance decreases for these models are shown in Figures 3 (d) and (e). The only exception is CHESS, which improved from Spider$\mathbb{M}$ to Spider$\mathbb{S}$, likely because its performance had already dropped considerably from Spider to Spider$\mathbb{M}$.

This led to an interesting finding: simply adding the words 'or multi' to the prompt caused a sharp decline in the model's ability to generate correct SQL, indicating that the model wasn't well-prepared for handling multiple SQL queries. Among these models, Llama-3.1-8B-Instruct and o3-mini were the most robust, with their EX scores dropping by only 0.007 and 0.015, respectively. In contrast, CHESS and Mistral-Large-Instruct-2411 experienced the largest performance drops, with declines of 0.292 and 0.217, respectively.

6.4 Challenges of Text-to-SQL Models in Supporting Multi-SQL Generation

Many text-to-SQL models struggle to accommodate Text-to-Multi-SQL, which is why Table 2 only reports the results of two text-to-SQL models. Firstly, traditional sequence-to-sequence models based on grammar decoders are incapable of generating multiple SQL statements. Secondly, even modern models built on LLMs often lack support for multi SQL in their design. For instance, the MAC-SQL (Wang et al., 2024) model breaks down the SQL generation task into a process of generating multiple SQL clauses, which are then combined to form a complete SQL statement. However, this generation process is not well-suited to the requirements of producing multiple SQL statements.

6.5 Analysis on DeepSeek-R1-Distill Models

The performance of the DeepSeek-R1-Distill models was relatively poor, so we conducted a detailed analysis of the generated data to identify the causes. First, we set the maximum generation token for these models to 1024, but around 17% of the examples exceeded this limit, which prevented the tasks from being completed. One reason for the token shortage is that the model repeatedly included the prompt's requirements in its output.

We randomly selected 30 examples where the task was completed successfully and analyzed the model's reasoning process. We found that only 11 of these examples were correct. The model made several mistakes, such as generating hallucinations, frequently using non-existent database columns, or selecting incorrect columns. It also seemed to lose track of the original schema structure as the reasoning progressed. Based on this analysis, we conclude that the Distill models used in this experiment are not well-suited for text-to-SQL tasks. Additionally, the influence of prior knowledge on the Distill models led to suboptimal performance, even after fine-tuning, compared to traditional models.

6.6 Analysis on Question Types

To explore the question types (see Appendix A) where the models perform poorly, and to identify the reasons for their lower performance, we selected three representative models and analyzed their execution accuracy across all question types. The results are shown in Figure 4. Overall, there are 14 question types in total, with type 14 having 4 subclasses to represent all patterns requiring 3 SQLs. This distinction was made to easily differentiate them from patterns requiring only 2 SQLs.

The fine-tuned Qwen2.5-Coder-7B-Instruct shows improvements across all question types. Type 12 sees marked improvement after fine-tuning, as the questions are actually quite simple; the previous low performance was due to the model's lack of training. XiYanSQL-QwenCoder-32B, while strong overall, struggles with multi-SQL tasks (types 14.1, 14.2, and 14.3), likely due to its prior single-SQL training. All models struggle with type 14.1, which is our most complex question type: combining one original question with one generated question that requires two SQL queries. Type 4 is also difficult for models, as it lacks common patterns and requires precise handling of each SQL.

7 Conclusion

We introduce the Text-to-Multi-SQL paradigm and the Spider$\mathbb{S}$ dataset, shedding light on key limitations in current Text-to-SQL models. Our experiments reveal that even state-of-the-art models like GPT-4o and XiYanSQL experience significant performance drops when tasked with multi-SQL generation. A factor is the models' sensitivity to prompt changes. To address these issues, future work should focus on improving model robustness and enhancing multi-query coordination to make data more accessible to non-technical users.

Limitations

Our work focuses solely on Text-to-SQL and Text-to-Multi-SQL, but even text-to-Multi-SQL cannot meet all user needs. As in the example in TAG (Biswal et al., 2024), users want to summarize the content of tables, and relying purely on SQL is currently difficult to achieve. Due to limitations in computing resources, we have not fine-tuned larger models (such as the 70B). At the moment, it is unclear how much better these models would perform after being fine-tuned.

References

Shanza Abbas, Muhammad Umair Khan, Scott Uk-Jin Lee, Asad Abbas, and Ali Kashif Bashir. 2022. A review of nlidb with deep learning: Findings, challenges and open issues. *IEEE Access*, 10:14927–14945.

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

Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E. Gonzalez, Carlos Guestrin, and Matei Zaharia. 2024. Text2sql is not enough: Unifying ai and databases with tag. *Preprint*, arXiv:2408.14717.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent advances in text-to-SQL: A survey of what we have and what we expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 2166–2187, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2024a. A preview of xiyan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv preprint arXiv:2411.08599*.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, et al. 2024b. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv preprint arXiv:2411.08599*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern,

Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz,

Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-generation database interfaces: A survey of llm-based text-to-sql. *Preprint*, arXiv:2406.08426.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Radu Cristian Alexandru Iacob, Florin Brad, Elena-Simona Apostol, Ciprian-Octavian Truică, Ionel Alexandru Hosu, and Traian Rebedea. 2020. Neural approaches for natural language interfaces to databases: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 381–395, Barcelona, Spain (Online). International Committee on Computational Linguistics.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023a. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023b. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

Peter Lawrence. 2024. Text-to-graph via llm: pre-training, prompting, or tuning.

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

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

Haoran Luo, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, et al. 2023. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *arXiv preprint arXiv:2310.08975*.

Rui Ma, Kai Zhang, Zhenying He, Yinan Jing, X Sean Wang, and Zhenqiang Chen. 2025. Chase: A native relational database for hybrid queries on structured and unstructured data. *arXiv preprint arXiv:2501.05006*.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan

Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *Preprint*, arXiv:2410.01943.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Preprint*, arXiv:2304.11015.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *arXiv preprint arXiv:2205.06983*.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *Preprint*, arXiv:2208.13629.

Qwen, :, 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, Jiaxi 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, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Cedric Renggli, Ihab F Ilyas, and Theodoros Rekatsinas. 2025. Fundamental challenges in evaluating text2sql solutions and detecting their limitations. *arXiv preprint arXiv:2501.18197*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

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

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*.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. Mac-sql: A multi-agent collaborative framework for text-to-sql. *Preprint*, arXiv:2312.11242.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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.

Weixu Zhang, Yifei Wang, Yuanfeng Song, Victor Junqiu Wei, Yuxing Tian, Yiyan Qi, Jonathan H. Chan, Raymond Chi-Wing Wong, and Haiqin Yang. 2024. Natural language interfaces for tabular data querying and visualization: A survey. *Preprint*, arXiv:2310.17894.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

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

Yuhang Zhou, Yu He, Siyu Tian, Yuchen Ni, Zhangyue Yin, Xiang Liu, Chuanjun Ji, Sen Liu, Xipeng Qiu, Guangnan Ye, et al. 2024. r3-nl2gql: A model coordination and knowledge graph alignment approach for nl2gql. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13679–13692.

Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. 2024. Large language model enhanced text-to-sql generation: A survey. *Preprint*, arXiv:2410.06011.

A Data Generation Patterns

To enhance the diversity and complexity of text-to-SQL datasets, we propose a systematic approach to data generation. Our methodology is based on modifying existing text-to-SQL pairs, ensuring that a single natural language query can correspond to multiple SQL queries while preserving the semantics of the original question. This approach allows us to construct a dataset with richer linguistic and structural variations, facilitating robust model training and evaluation.

We adopt multiple patterns to transform single-query data into multi-query pairs. These patterns fall into 14 different categories, each designed to introduce different types of complexity while maintaining the integrity of the original intent. Below is a detailed introduction of our generation patterns.

1. **Reverse Selection in One-to-Many Relationships**

   To introduce compositional complexity into text-to-SQL pairs, we employ reverse selection on one-to-many relationships. This approach ensures that a given natural language query requires multiple SQL statements to retrieve the requested information, as opposed to a single SQL query.

   In relational databases, a one-to-many relationship exists when an entity in one table (e.g., departments) is associated with multiple entities in another table (e.g., teachers). The department table serves as the parent (one), while the teacher table acts as the child (many), with the foreign key in the teacher table referencing the primary key in the department table.

   Consider the following example, where the original query is:

   Find the names of teachers who are older than 40.

This query can be answered using a single SQL statement:

SELECT name FROM teacher WHERE age > 40;

To enforce multi-query execution, we introduce a reverse selection constraint by modifying the query as follows:

Find the names of teachers who are older than 40 and the departments of the remaining teachers.

Here, the teacher name and the department name of other teachers are semantically incompatible in a single SQL query, necessitating at least two queries:

SELECT name FROM teacher WHERE age > 40;

SELECT department_name FROM teacher WHERE age <= 40;

This transformation ensures that a single query is insufficient to retrieve all requested information, thereby enriching the complexity of text-to-Multiple-SQLs pairs.

2. **Reverse Selection in One-to-One Relationships**

   Unlike the one-to-many relationships discussed previously, a one-to-one relationship occurs when each record in a table corresponds to a single record in another table. In this case, reverse selection cannot involve direct inclusion of attributes from the same table, as such queries can always be answered using a single SQL statement. Therefore, to enforce compositional complexity, reverse selection in one-to-one relationships must ensure that the two queries retrieve different types of information.

   A critical constraint of one-to-one reverse selection is that the secondary query must retrieve different attributes. If the modified query asks for the same attribute from different records, it can always be rewritten as a single SQL query, rendering the transformation invalid. So we need to ensure the queries ask for logically distinct attributes. Consider the following transformation:

   Find the names of teachers who are older than 40 and retrieve the average age of the remaining teachers.

   This results in the following SQL queries:

   SELECT name FROM teacher WHERE age > 40;

   SELECT AVG(age) FROM teacher WHERE age <= 40;

   Here, the first query retrieves individual teacher names, while the second query retrieves an aggregate statistic, ensuring they cannot be merged into a single SQL statement.

3. **Reverse Selection in One-to-One Relationships Without Aggregation**

   One-to-one reverse selection can be further constrained by eliminating aggregation functions, making the compositional SQL queries more structurally distinct while still requiring multiple SQL queries to fully answer the question. This ensures that query decomposition is necessary but does not rely on aggregate statistics to enforce multi-query generation.

4. **Distinguishing Separate Entities for Independent Queries**

   In the process of constructing diverse SQL queries, one effective approach is to generate questions that target distinct entities within the database. At first glance, it may seem intuitive that retrieving different types of information always necessitates multiple SQL queries. However, this is not necessarily the case. When two entities are inherently linked through a foreign key relationship, it is often possible to retrieve both pieces of information using a single query. For instance, if the original query asks for the names of all teachers, it can be extended to also request their department names without requiring an additional SQL statement. Since teachers are linked to departments via a foreign key, a simple join operation suffices to bring both sets of information together within a single query.

   However, to ensure that two distinct queries are required rather than one combined query, the entities being retrieved must not only belong to different tables but also be subject to independent constraints. Consider the case where we modify the original question from "Find the names of teachers older than 40" to "Find the names of teachers older than 40 and the names of departments with more than 40 members." This slight modification forces the

retrieval of two independent sets of information—one about teachers filtered by age, and another about departments filtered by the number of members. Since these conditions apply to separate tables and do not share a common join path that would allow them to be merged into a single SQL query, it becomes necessary to generate two distinct SQL statements.

Find the names of teachers who are older than 40 and the names of departments with more than 40 people.

This question introduces conditions on different attributes from separate tables, making it impossible to construct a single SQL query that answers both parts. Instead, two independent queries must be generated:

SELECT name FROM teacher WHERE age > 40;

SELECT name FROM department WHERE num_people > 40;

5. **Asymmetric Query Expansion: One Condition, One Open Query**

   In contrast to the previous method, which ensured that two distinct objects were queried with their own independent constraints, this method introduces an asymmetry: one of the SQL queries retains a specific condition while the other remains unrestricted. This subtle shift in design changes the way queries are structured and how they interact with the database schema. Instead of ensuring that both queries are filtered by independent constraints, one query maintains a filtering condition while the other remains a general retrieval of all records from a related table.

   To illustrate, consider an original query asking for "Find the names of all teachers." A straightforward extension would be to also retrieve the names of all departments, a task that could easily be achieved using a single SQL query through a simple join operation. However, if we modify this to "Find the names of teachers who are older than 40 and all department names," we introduce a structural asymmetry—one query applies a condition (teachers older than 40), while the other query retrieves all departments without filtering. This distinction ensures that the two queries must remain separate rather than being merged into a single SQL statement.

   The process of generating such queries relies on selecting an appropriate example from existing data, ensuring that one query retains a condition while the other does not. The system systematically scans the dataset to identify candidate queries that fit this asymmetric pattern, filtering out those that contain multiple constraints or already require multiple SQL statements. Once an appropriate pair of queries is selected, the prompt is carefully constructed to emphasize the independence of the two queries while maintaining clarity in natural language.

6. **Blending Aggregates with Individual Attributes: Merging Statistical Insights with Specifics**

   In natural language database queries, there is often a need to balance between querying individual records and summarizing trends. A common way to expand a query is by introducing an aggregation function alongside a standard selection. This method ensures that both specific details and overarching insights are retrieved simultaneously, making the query richer and more informative.

   For example, consider the original query:

   Find the names of teachers who are older than 40.

   This query retrieves a list of individual teachers who meet the condition, focusing on granular details. However, by expanding the question to include an aggregation function, we can introduce a statistical perspective. The modified query becomes:

   Find the names of teachers who are older than 40 and their average age.

   Here, in addition to retrieving individual teacher names, the query also calculates the overall average age of teachers above 40. This transformation creates a dual perspective—while one part of the query extracts direct information from the database, the other part summarizes the information using an aggregation function.

   To construct such queries correctly, it is crucial to ensure that both components operate

on the same filtered dataset. This prevents inconsistencies, such as computing an average age across all teachers instead of just those older than 40. The corresponding SQL query would look like this:

SELECT name FROM teachers WHERE age > 40;

SELECT AVG(age) FROM teachers WHERE age > 40;

Alternatively, a single SQL statement can often achieve the same result using a GROUP BY clause:

SELECT name, (SELECT AVG(age) FROM teachers WHERE age > 40) AS avg_age FROM teachers WHERE age > 40;

Since this data generation method can also be completed with a SQL query, we introduced the sql_0 statement to store all queries integrated into one SQL.

7. **Querying Different Grouping Dimensions**

   Grouping data is a fundamental aspect of SQL queries, allowing us to derive meaningful insights by categorizing and summarizing records. While most queries focus on a single way to group data, an alternative approach introduces multiple GROUP BY clauses, effectively broadening the scope of the query. By shifting the lens through which data is aggregated, we can uncover deeper relationships within the dataset that a single GROUP BY might miss.

   This method is especially useful when dealing with datasets where different categorical attributes can provide valuable insights when analyzed separately. For example, consider a dataset of department management. The original question:

   Tell me which states have at least 3 heads born there?

   The corresponding SQL query would look like this:

   SELECT born_state FROM head GROUP BY born_state HAVING COUNT(*) >= 3;

   However, what if we want to explore a different way of grouping the data? We could also ask:

   Tell me which states have at least 3 heads born there and which age groups have at least 3 heads?

   The result is that we have to use 2 different SQL queries to answer this question:

   SELECT age FROM head GROUP BY age HAVING COUNT(*) >= 3;

   SELECT born_state FROM head GROUP BY born_state HAVING COUNT(*) >= 3;

8. **Switching Between HAVING and LIMIT for Diverse Filtering**

   This method modifies SQL queries by ensuring that one query filters aggregated results using HAVING, while the other applies ORDER BY and LIMIT. Unlike the previous approach, which changed the GROUP BY column to introduce variation, this method focuses on switching between filtering patterns to generate distinct yet complementary queries.

   If the original query includes LIMIT, the second query must replace it with a HAVING clause to introduce a different filtering method. Conversely, if the original query already uses HAVING, the new query must avoid it and instead adopt an ORDER BY and LIMIT structure. This ensures that the two queries cannot be merged into one, as they use fundamentally different ways of selecting data.

   For example, if the original query retrieves the department with the most teachers using ORDER BY and LIMIT:

   SELECT department FROM teachers GROUP BY department ORDER BY COUNT(*) DESC LIMIT 1;

   The second query should instead filter using HAVING, such as selecting age groups where the number of teachers exceeds a threshold:

   SELECT age FROM teachers GROUP BY age HAVING COUNT(*) > 2;

   This variation ensures that both queries serve different analytical purposes while maintaining structural diversity. By alternating between HAVING and LIMIT, we create queries that offer different perspectives on the dataset, preventing them from being collapsed into a single SQL statement.

9. **Reordering the Same Data from Different Perspectives**

This method transforms a query that sorts data using ORDER BY into two separate queries, each sorting the same dataset by a different attribute. Unlike cases where LIMIT is involved, which restricts the number of results, this approach ensures that both queries return the full dataset, just presented in different orders.

Consider the original query:

Find the names of all teachers, sorted by age in descending order.

Original SQL Query:

SELECT name FROM teachers ORDER BY age DESC;

To introduce variety, we modify the query by adding an alternative sorting criterion. Instead of only ordering by age, we ask for two separate orderings—one by age and another by salary. The question can be transformed into:

Give me two separate lists of teacher names: one sorted by age in descending order and the other sorted by salary in descending order.

Corresponding SQL Queries:

SELECT name FROM teachers ORDER BY age DESC;

SELECT name FROM teachers ORDER BY salary DESC;

This transformation ensures that both results remain distinct and cannot be merged into a single SQL query.

10. **Sorting One Object While Leaving the Other Unordered**

    This method builds upon the previous approach of modifying ORDER BY queries but introduces a crucial difference: instead of applying sorting criteria to both SQL queries, one query retains the ORDER BY clause while the other remains unsorted. The key idea is to introduce contrast—one dataset follows an explicit order while the other remains in its natural or default arrangement.

    Consider the original query:

    Find the names of all teachers, sorted by age in descending order.

    Original SQL Query:

    SELECT name FROM teachers ORDER BY age DESC;

    To introduce variation while maintaining meaningful structure, we modify the query to introduce an additional unordered dataset. Instead of just listing teachers sorted by experience, we now also retrieve the departments they belong to, but without any explicit sorting applied to them.

    Modified Question:

    Find the names of all teachers, sorted by their ages in descending order, and separately list all department names.

    Corresponding SQL Queries:

    SELECT name FROM teachers ORDER BY age DESC;

    SELECT department_name FROM departments;

    This modification ensures that two separate results are produced, but unlike the previous approach (where both queries applied sorting in different ways), here only one result is ordered while the other remains unaffected. This distinction is important because it emphasizes that ORDER BY is not necessarily required for every SQL statement—it depends on the nature of the query and what needs to be compared or presented.

11. **Sorting One Object with LIMIT Condition While Leaving the Other Unordered**

    This method is quite similar to the previous method (10) in that it requires an SQL query with an ORDER BY clause and an extra query remaining unsorted. However, the ORDER BY query also includes an additional LIMIT condition. Furthermore, we intentionally select the additional SQL query to include a WHERE clause while minimizing the use of any aggregation functions. This is set to make the problem more challenging and to ensure the conditions of both 2 queries are clearly described to avoid any potential ambiguity.

    Consider the original query:

    Find the name of the oldest teacher

    Original SQL Query:

    SELECT name FROM teachers ORDER BY age DESC LIMIT 1;

    To introduce variation while maintaining meaningful structure, we modify the query to

introduce an additional unordered query with a WHERE clause. We intentionally select an additional query with a WHERE clause and avoid using any aggregation function.

Modified Question:

Find the name of the oldest teacher and the ages of all teachers whose names contain the letter "A".

Corresponding SQL Queries:

SELECT name FROM teachers ORDER BY age DESC LIMIT 1;

SELECT age FROM teachers WHERE name LIKE "%A%";

This modification is similar to the previous method, but introduces more conditions: One query is required to select only some results from the whole sorted list, while another query should return the unsorted results under the WHERE condition. This type of modification further challenges agents' ability to comprehend the nature of the questions.

12. **Dual Attribute Selection Sorted by the Same Attribute but in Reverse Orders with LIMIT Condition**

    This method also requires an original SQL query to return results constrained by the LIMIT clause sorted by some attributes. We then add another query to select another attribute sorted by exactly the same attribute but in reverse order. Notice that the results are still constrained by the LIMIT clause. Intuitively, one simple way to achieve this type of modification is to transform the statements in the original question that use superlative forms into their opposites (e.g., changing 'maximum' to 'minimum' and 'most' to 'least'). Also, we intentionally include some questions to change the number of results required by the LIMIT clause to make this type of modifications more variant and more challenging.

    Consider the original query:

    Show the name of the teacher with the highest salary.

    Original SQL Query:

    SELECT name FROM teachers ORDER BY salary DESC LIMIT 1;

    Based on the the detailed explained pattern above, we introduce an additional query to select another attribute in the same table sorted by exactly the same attribute (e.g, salary) but in the reverse order (e.g, ASC). We can also change the number of results returned by changing the LIMIT clause (e.g, the lowest three salaries).

    Modified Question:

    Show the name of the teacher with the highest salary and the office numbers of the teachers who have the lowest three salaries.

    Corresponding SQL Queries:

    SELECT name FROM teachers ORDER BY salary DESC LIMIT 1;

    SELECT office_no FROM teachers ORDER BY salary ASC LIMIT 3;

    This modification requires the agent to output two distinct SQL queries to select two subsets of different attributes based on the sorted result of the same attribute but in reverse orders. This type of modification could pose challenges for some agents, as they must select two distinct attributes based on a condition and its nearly opposite form.

13. **Dual Attribute Selection Sorted by the GROUP BY clause of the Primary Key but in Reverse Orders with LIMIT Condition**

    This method can be considered as a more challenging variant of the previous method (12) as the main logic is quite similar. The only difference is that we use the GROUP BY clause of the primary key of the table to do sorting (e.g, GROUP BY id ORDER BY COUNT(*) DESC LIMIT 1). This task is considered more challenging as the original question requires the agent to understand the structure of the table and use the GROUP BY clause properly. A more complex task in the same setting to select two attributes sorted by some attributes of groups in reverse orders can result in greater confusion for agents.

    Consider the original query:

    Show the status of the city that has hosted the greatest number of competitions.

    Original SQL Query:

    SELECT T1.Status FROM city AS T1 JOIN farm_competition AS T2 ON

T1.City_ID = T2.Host_city_ID GROUP BY T2.Host_city_ID ORDER BY COUNT(*) DESC LIMIT 1;

Based on the additional explanation of this variant, we select another proper attribute (e.g, the primary key itself) in the table and only changes the sorting order to the reverse one.

Modified Question:

Show the status of the city that has hosted the greatest number of competitions and the city ID with the fewest competitions.

Corresponding SQL Queries:

SELECT T1.Status FROM city AS T1 JOIN farm_competition AS T2 ON T1.City_ID = T2.Host_city_ID GROUP BY T2.Host_city_ID ORDER BY COUNT(*) DESC LIMIT 1;

SELECT T1.City_ID FROM city AS T1 JOIN farm_competition AS T2 ON T1.City_ID = T2.Host_city_ID GROUP BY T2.Host_city_ID ORDER BY COUNT(*) ASC LIMIT 1;

14. **Question Generation with Three SQL Queries**

We also introduce a small proportion of questions that require three SQL queries to address. These questions are generated based on some of the previous methods. As this could be a more challenging and difficult task for agents, we only select some relatively simple methods to generate questions.

14.1 **Combining One Original Question with One Already Generated Question Requiring Two SQL Queries**

An intuitive way to generate questions requiring three SQL queries to address is to combine one original question from the dataset with one generated and well-evaluated question requiring two queries. To ensure that the final generated question is reasonable, relatively concise, and unambiguous, we intentionally select those original SQL queries that include a WHERE clause while excluding certain keywords that may form complex questions (such as LIMIT, GROUP, EXCEPT, INTERSECT, UNION, ORDER) and control the length of the selected SQL queries. Then we sample a well-evaluated question from our generated dataset and ask the agent to merge the two questions. The agent is also suggested to use some keywords like "separately" or "first, then," to avoid any potential ambiguous statements.

14.2 **Querying Different Grouping Dimensions to Generate three queries**

This method is quite similar to the previous method 7 as we still ask to group the table data based on different attributes separately, but now three different attributes are used instead of only two.

Two example generated questions are shown below:

1. *Show the number of teachers for each department, age, and country separately.*
2. *Find the department, the age group and the country group with the highest number of teachers.*

14.3 **Reordering the Same Data from Three Different Perspectives**

This method is quite similar to the previous method 9 as we still ask to provide lists of one single attribute sorted separately by different attributes, but this time we require three lists. Notice that each query sorted by each attribute can still be in ascending or descending order randomly.

An example generated questions are shown below:

Provide me with three lists of teacher names, sorted by age, department id and salary in descending order, respectively.

14.4 **Two or Three Queries with DISTINCT**

Previously our methods didn't consider the keyword DISTINCT to avoid any potential ambiguous statement. Here we ask the agent to generate a small subset of questions to ask for distinct values for selected columns only. The generated questions may require two or three queries to return only distinct/unique values for mentioned attributes.

A simple example question is shown below:

Provide me with distinct teacher name, unique department names, and distinct teacher ages, respectively.

B Implementation Details

All experiments were conducted using LLAMA Factory (Zheng et al., 2024), with LoRA-based (Hu et al., 2021) fine-tuning utilizing the bf16 precision. The relevant hyperparameters were configured as follows:

- preprocessing_num_workers: 16
- per_device_train_batch_size: 1
- gradient_accumulation_steps: 4
- learning_rate: 1.0e-4
- num_train_epochs: 3.0
- lr_scheduler_type: cosine
- warmup_ratio: 0.1

Other parameters were set to the default values provided by LLAMA Factory (Zheng et al., 2024). The experiments were conducted on single or multiple A100 GPUs (40GB/80GB). For multi-GPU setups, DeepSpeed was used to optimize parallel training.